LEARNING TO WALK USING DEEP REINFORCEMENT LEARNING AND TRANSFER LEARNING

A Thesis
Presented to
The Academic Faculty

By

Wenhao Yu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

August 2020

LEARNING TO WALK USING DEEP REINFORCEMENT LEARNING AND TRANSFER LEARNING

Approved by:

Professor Greg Turk, Advisor
School of Interactive Computing
Georgia Institute of Technology

Professor C. Karen Liu, Advisor
School of Engineering
Stanford University

Professor Charlie C. Kemp
Wallace H. Coulter Department of Biomedical Engineering
Georgia Institute of Technology

Professor Sergey Levine
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Professor Michiel van de Panne
Department of Computer Science
University of British Columbia

Date Approved: April 28, 2020

ACKNOWLEDGEMENTS

First, I want to thank my wife, Mengmeng, for your unconditional support and love throughout my PhD program. I am deeply grateful for your patience, encouragement, and love during this journey that is full of joy and setbacks. I cannot imagine going through the ups and downs in the past five years without you alongside.

I also want to thank my parents, Xiulin and Chenghe, for guiding me through my life and supporting me in pursuing what I love. It is you who shaped me into who I am today and I am deeply thankful for that. I would also like to thank my in-laws, Yongfeng and Jun, for encouraging me to pursue my degree and for sharing their wisdom with me.

I want to thank the members of my committee: Charlie Kemp, Sergey Levine, Michiel van de Panne, as well as my qualifier committee member James Hays. Each of you encouraged and helped me become a stronger and solid researcher with insightful suggestions on my research and presentation skills. You have also acted as role models that I look up to and hope to become in my future academic career. In particular, I want to thank Charlie for the valuable advice during my study. You taught me how to be more rigorous and strive for big impacts in my research and I am grateful for all the interactions we have had.

I wish to thank my lab-mates and friends during the program: Jie Tan, Yunfei Bai, Sehoon Ha, Ariel Kapusta, Alex Clegg, Mukul Sati, Visak Kumar, Ashish Gupta, Kelsey Kurzeja, Yang Tian, Zackory Erickson, Henry Clever, Yunbo Zhang, Nitish Sontakke, Maksim Sorokin, Xiangyu Li, Xinyan Yan, and many more. Thank you for your helpful discussions with me about my research and my life. I want to especially thank Jie and Yunfei for your guidance to establish my research direction, your insightful suggestions during my study, and your advice during my last internship. I also want to thank Yuting Ye, Bryan Klimt, Topraj Gurung, Erwin Coumans, Tingnan Zhang, and Wenlong Lu for the advice and help during my past internships.

Finally, I want to express my deepest gratitude to my advisors, Karen Liu and Greg

Turk. I feel extremely fortunate to have you as my advisors during this fantastic and fulfilling journey. You motivated me to find the research direction I like, to take up challenging problems, and provided me the freedom to explore my ideas and to become an independent researcher. You demonstrated to me how fun and rewarding doing research can be and how the quest for knowledge is not simply a job duty as a scientist, but also an enjoyable journey for life. You were my advisors, colleagues, and friends and I cannot image how I would have become who I am now without your guidance throughout my PhD.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\mathcal{A}$ The action space of characters or robots. 18, 19, 34

$\mathbf{a}$ Actions applied to characters or robots. 18–21, 26, 46–50, 54, 57, 61, 64, 75, 76, 102

$\boldsymbol{\mu}$ A vector of physical parameters defining the dynamic model (e.g. friction coefficient). 18, 46–53, 56–62, 71–75, 79–81, 83, 85, 86

$\mathcal{O}$ The observation space of characters or robots. 18

$\mathbf{o}$ Observation from simulated characters or robots. 22, 34, 46–50, 54, 61, 64, 75, 76, 85, 101, 102

$\mathcal{P}$ Transition function of MDP. 18, 19, 46, 48–50

p_0 Initial state distribution. 18, 49

$\mathcal{R}$ Reward function of MDP. 18, 49, 52, 54, 55, 78, 87

$\mathcal{S}$ The state space of characters or robots. 18, 19, 34

$\mathbf{s}$ State of simulated characters or robots. 18–21, 26, 46, 49, 50, 52, 54, 55, 57, 100

$\mathbf{x}$ A vector representing the level of assistance used during the curriculum for learning locomotion.). 27–33

SUMMARY

We seek to develop computational tools to reproduce the locomotion of humans and animals in complex and unpredictable environments. Such tools can have significant impact in computer graphics, robotics, machine learning, and biomechanics. However, there are two main hurdles in achieving this goal. First, synthesizing a successful locomotion policy requires precise control of a high-dimensional under-actuated system and striking a balance among a set of conflicting goals such as walking forward, energy efficiency, and keeping balance. Second, the synthesized locomotion policy needs to generalize to new environments that were not present during optimization and training in order to cope with novel situations during execution. In this thesis, we introduce a set of learning-based algorithms to tackle these challenges and make progress towards achieving automated and generalizable motor learning. We demonstrate our methods on training simulated characters and robots to learn locomotion skills without using motion data, and on transferring the simulation-trained locomotion controllers to real robotic platforms.

We first introduce a Deep Reinforcement Learning (DRL) approach for learning locomotion controllers for simulated legged creatures without using motion data. We propose a loss term in DRL objective that encourages the agent to exhibit symmetric behavior and a curriculum learning approach that provides modulated physical assistance in order to achieve successful training of energy-efficient controllers. We demonstrate the results of this approach across a variety of simulated characters that, when we combine the two proposed ideas, achieve low-energy and symmetric locomotion gaits that are closer to those seen in real animals than alternative DRL methods.

Next, we introduce a set of Transfer Learning (TL) algorithms that generalize the learned locomotion controllers to novel environments. Specifically, we focus on the problem of transferring a simulation-trained locomotion controller to a real legged robot, also known as the Sim-to-Real transfer problem. Addressing the Sim-to-Real transfer prob-

lem would allow robots to leverage the modern machine learning algorithms and compute power in learning complex motor skills in a safe and efficient fashion. However, this is also a challenging problem because the real-world is noisy and unpredictable. Within this context, we first introduce a transfer learning algorithm that can successfully operate in unknown and changing dynamics within the training dynamics. To allow successful transfer outside the training environments, we further propose an algorithm that uses a limited amount of samples in the testing environments to adapt the simulation-trained policy. We demonstrate two variants of the algorithm that were applied to achieve Sim-to-Real transfer for a biped robot, Robotis Darwin OP2, and a quadruped robot, Ghost Robotics Minitaur, respectively.

Finally, we consider the problem of safety during policy execution and transfer. We propose the training of a universal safe policy (USP) that controls the robot to avoid unsafe states from a diverse set of states, and an algorithm to combine a USP and a task policy to complete the task while acting safely. We demonstrate that the resulting algorithm can allow policies to adapt to notably different simulated dynamics with at most two failure trials, suggesting a promising path towards learning robust and safe control policies for sim-to-real transfer.

CHAPTER 1
INTRODUCTION

The evolution of physical forms, motor controllers, and environments have led to an incredibly diverse and agile set of locomotion gaits in different animals: we see humming birds hovering in the mid-air by flapping their wings at 70 Hz, creating a whirring sounds audible to humans, cheetahs sprinting to chase the preys with an acceleration comparable to a supercar, and mountain goats climbing near-vertical cliffs. Each distinct locomotion gait produces a coordinated, rhythmic movement that interacts with the environment through contact, drag, or thrust forces to transport the center of mass while striking an intricate balance between speed, energy efficiency, stability, endurance, and maneuverability.

Understanding and reproducing how animals acquire and produce these elegant movements has been a long-standing interest for researchers in computer graphics, robotics, biomechanics, and artificial intelligence. Research in this area can lead to wide impacts on various aspects of human society. For example, being able to create virtual humans and animals that move like their natural counterparts in an automatic way would allow more realistic and interactive characters in computer games, movies, and virtual reality applications. In addition, equipping robots with animal-like locomotion skills would allow them to automate tasks that involve intense human labors like parcel delivery [1], to perform tasks that are dangerous to humans such as disaster relief [2], and to carry out scientific exploration tasks like outer space exploration [3]. Moreover, understanding how humans walk or run can help design exoskeleton systems to help people in need of assistance achieve independence and to engage people in physical activities, improving general health [4].

The last few decades have seen tremendous progress in creating realistic movements for virtual characters and real robots. For example, we see virtual animals that are almost indistinguishable from their real-world counterparts in movies like Jungle Book and The

Lion King. In robotics, we see robots like Boston Dynamics Atlas that can perform astonishing movements such as back-flips and parkour. Despite the notable achievements in these fields, existing systems for authoring these complex movements for animated characters and real robots still involves a large amount of manual efforts and prior knowledge. For example, creating a successful acrobatic movement for Atlas can take hundreds of trials and manual adjustments spanning over several months. Moreover, this tedious procedure needs to be repeated for every new environment and motion due to the lack of generalization. **Can we synthesize these compelling motions in an automatic and generalizable way?**

For virtual characters, a natural-looking motion must satisfy the laws of physics. For example, a terrestrial animal-like character should only receive external forces through gravity and contact forces. This can be addressed by the use of physics-based simulation. For characters that resemble humans or legged animals, they are usually represented as multiple rigid bodies connected through joints, which can exert internal torques to achieve the target motion. Techniques for simulating such characters are well developed and there exists a variety of simulation tools that allows researchers and artists to model physics-based characters [5, 6, 7].

However, controlling these simulated characters to automatically acquire animal-like movements in a simulated environment presents several challenges. To begin with, understanding the underlying reward that shapes the locomotion gaits is non-trivial. Though researchers have proposed several hypothesis that the animals locomotion skills are optimized for, such as energy efficiency, speed, stability, etc [8], factors like emotion of the creature will have noticeable effects on the resulting motion [9], but are not straightforward to incorporate in the optimization problem. Even if we limit our quest to the objective terms that are well-defined, finding an optimal trajectory for these goals is still difficult. This is due to the large search space that the optimizer needs to search over, as well as the complex relation between the control signal (controlling torques) and the resulting motion, governed

by the nonlinear dynamical equations. As a result, most of the prior research simplifies the problem by introducing prior knowledge or external datasets that are specific to the task of interest, such as finite state machines, or motion capture data. This makes formulating and solving the optimization problem more manageable, but requires additional manual efforts and leads to less automatic algorithms. Furthermore, they are not applicable to creatures and tasks where motion data is not available. In this thesis, we develop algorithms for synthesizing compelling locomotion animations directly from the simple and general principles from biomechanics, without using external data or prior knowledge specific to the character and the motion.

Synthesizing locomotion controllers for real physical robots poses additional challenges beyond the ones for virtual characters. First, the real world is noisy and cannot be manipulated: unlike computer simulated environments, we cannot get the precise state of the robot at every moment or reset the robot to arbitrary positions and velocities. Consequently, the same control commands can lead to different resulting motions on the real hardware. Second, performing an extensive amount of trials on the real robot is expensive, human labor intensive, and potentially dangerous. As a result, many algorithms developed for synthesizing locomotion controllers for simulated characters are not feasible on real robots.

One promising direction to address these challenges is to model the real robot in a physically simulated environment, find an optimal controller or trajectory in the simulation, and then apply them to the real robot. However, existing methods for synthesizing locomotion controllers are usually limited to their training environments and do not generalize to new situations. As a result, control policies trained in a simulated environment usually do not work directly on the real robot due to the discrepancy between the simulation and the real-world, also known as the Reality Gap [10]. Overcoming the reality gap requires the development of novel learning algorithms that allows policies to generalize beyond their training environments. In this thesis, we propose a set of transfer learning algorithms that make progress towards this goal and achieve successful sim-to-real transfer for locomotion

policies on two legged robots.

1.1 Thesis Overview

In this dissertation, we present a set of computational tools to achieve automatic and generalizable locomotion learning for simulated characters and real, physical robots. In Chapter 2, we survey the related works in the field of character animation, robotics, and machine learning. In Chapter 3, we provide the mathematical backgrounds for formulating the motion synthesis problem and reviewed a few reinforcement learning algorithms that we will be using to solve the motor control problems. We then introduce a learning-based algorithm for creating symmetric and low-energy locomotion controllers for various legged characters (Chapter 4). Next, we investigate how to transfer the simulation-trained locomotion policies to significantly different environments, including the real world (Chapter 5). In Chapter 6, we investigate the problem of training a safety-aware control policy for safe exploration and execution during learning and transfer. We conclude the thesis with discussions of the proposed methods, and suggestions of future work directions (Chapter 7).

1.1.1 Learning Legged Locomotion with Deep Reinforcement Learning

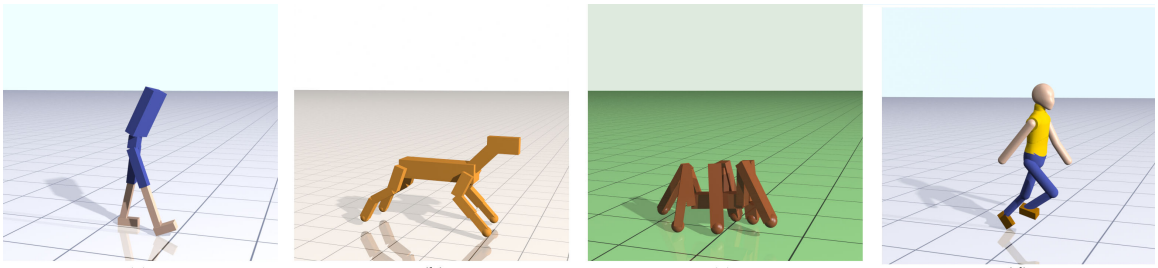


Figure 1.1: Locomotion controller trained for different creatures.

Creating an animated character or a robot that can walk like a real human or animal is a long-standing challenge in computer graphics, robotics and machine learning. Due to the

inherent instability of legged creatures, the high dimensional action space, and the discontinuous dynamics caused by contacts, synthesizing a policy that controls the character to move forward is very challenging. As a result, many previous methods rely on motion capture data, or breaking down the locomotion gait cycle to guide the optimization to achieve the desired motion.

Recent developments in Deep Reinforcement Learning (DRL) have demonstrated training of locomotion controllers using simple and generic principles from biomechanics. However, the resulting motions usually do not look realistic due to two reasons. First, the motions appear much more energetic than biological systems. Second, an agent with perfectly symmetrical morphology often produces visibly asymmetrical motion. In Chapter 4, we present a deep reinforcement learning-based algorithm that addresses these issues and produces symmetric and low-energy locomotion controllers [11].

Minimizing energy consumption in a DRL setting involves careful selection of the weights for the reward term: penalizing energy consumption too aggressively would lead to insufficient exploration, while penalizing energy too little would lead to energetic motion. Inspired by physical learning assistance such as the training wheels for riding a bicycle, we develop a curriculum learning algorithm that applies external forces to assist the character in learning to walk or run, and then gradually remove the assistance as it gets better at the task until the controller can work without assistance. We demonstrate that this scheme allows us to reliably synthesize locomotion controllers that use low energy consumption without tedious tuning of the reward function.

Minimizing energy consumption alone is not sufficient to ensure natural movement: the character may still develop a smooth but asymmetric locomotion gait. Though it is possible to constrain the motion to be symmetric, designing this constraint would require us to know the frequency of the motion, and can be difficult for complex characters with many appendages. Instead, we propose a mirror symmetry loss that enforces the policy to be symmetric. We show that by doing this, we can obtain symmetric locomotion gaits,

while retaining the flexibility to generate asymmetric motion when desired. Our method is demonstrated on training locomotion controllers for a variety of simulated characters (Figure 1.1).

1.1.2 Sim-to-Real Transfer of Locomotion Policies

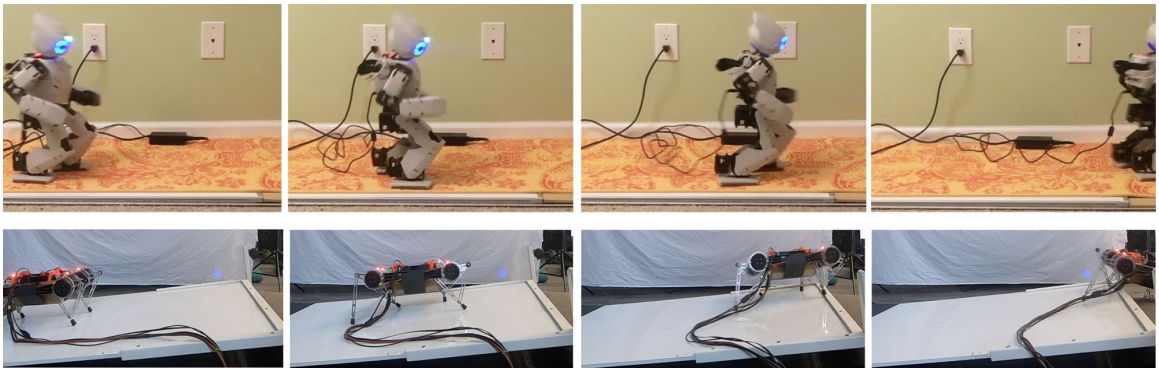


Figure 1.2: Robots learn to walk by transferring simulation-trained locomotion policies to the real-world.

Recent progress in DRL sheds light on developing complex motor skills in challenging situations. However, the policies found by DRL algorithms are usually limited to a single scenario and may not work if the target environment changes notably. A promising direction to address this issue is to use the idea of transfer learning which learns a model in a source environment and transfers it to a target environment of interest. Researchers have studied transfer learning for DRL in order to transfer a trained policy to different tasks, environments, and input domains. In this dissertation, we focus on the problem of transferring policies to novel environments, e.g. different physical properties, tilted ground, modeling error, etc. One notable example of transfer learning problems in this category is to transfer simulation-trained policies to real robots, i.e. Sim-to-Real transfer. Achieving successful Sim-to-Real transfer would allow robots to acquire complex motor skills in a safe and efficient way, and enable us to leverage the modern machine learning techniques and compute power.

In Chapter 5, we introduce a series of transfer learning algorithms that leads to suc-

cessful sim-to-real transfer for a biped and a quadruped robots [12, 13, 14, 15]. The core idea of our approach is to aggressively explore the "virtual world" through physical simulation and precompute many of the possible situations the robot might encounter when operating in testing environments. When deploying the trained policies in the testing environment, e.g. the real world, we identify the best policy to use from the family of policies trained in simulation. We investigate three ways to precompute the policies for a variety of situations in simulation, and two metrics for selecting the 'best' policy to be deployed in the testing environment. We find that different choices of these components would lead to transfer learning algorithms that are suitable for different types of discrepancies between training and testing environments and varied amount of real-world data requirement during the transfer. We demonstrate two combinations (described in Section 5.4 and Section 5.5) that achieve successful sim-to-real transfer for a biped robot, Robotis Darwin OP2, and a quadruped robot, Ghost Robotics Minitaur, as shown in Figure 1.2.

1.1.3 Training Safety-aware Locomotion Controllers

In order to survive unpredictable and ever-changing environments, animals have developed remarkable skills in identifying risks. For example, rodents have developed a specialized threat-detection system for identifying visual curs or odor of predators [16]. On the other hand, robotics controllers obtained by DRL algorithms are typically trained to be aggressive in optimizing the task rewards such as moving forward, or lifting an object, without considering the potential safety issues such as the robot falling down, or breaking other objects. This limits the learned policies from being applied to safety-critic problems. Enforcing safety in DRL algorithms has recently been an area of interest in the robotics and machine learning community. Most existing work in safe DRL focus on ensuring safety during the training process, assuming that the training process happens entirely in the target environment. In this dissertation, we investigate a different problem setting for safe DRL: given a trained control policy in a source environment, such as simulation, can we

develop algorithms to help it achieve safe deployment in a target environment such as the real-world?

In Chapter 6, we introduce a model-free Deep Reinforcement Learning-based approach for safely transferring simulation-trained control policies to notably different simulation environments without querying the testing environment. The core observation we make is that a policy trained to remain safe is more transferable than a policy trained to optimize a particular task. As a result, if we can train a separate policy dedicated to keep the robot within the safety regions and be able to intelligently select which policy to query for actions at each timestep, we can achieve more transferable and safe policies. To this end, we propose to train a Universal Safe Policy (USP), as well as a safety critic model for selecting when to use the USP model. We demonstrate that combining the task policy and USP enables policies to be transferred to environments that are significantly different from the training environments.

1.2 Contributions

The research work described in this thesis make several contributions to the community of computer graphics, robotics, and machine learning, as listed below.

A minimalist approach to learning of locomotion gaits. We present a novel curriculum learning algorithm and mirror symmetry loss for synthesizing locomotion controllers that are low-energy and symmetric. By applying the curriculum learning algorithm, we are able to solve difficult policy optimization problems where vanilla DRL algorithms would fail, and achieve locomotion policies that are energy efficient. Furthermore, by augmenting the DRL objective with the proposed mirror symmetry loss, we encourage the policy to obtain symmetric gaits when symmetry is beneficial. Different from imitation learning approaches that depend on high quality motion examples, our locomotion learning algorithm, combining low-energy and symmetry, produces believable locomotion gaits for a variety of simulated characters without utilizing motion data or gait analysis.

A series of transfer learning algorithms for transferring simulation-trained locomotion policies to real legged robots. We introduce a series of transfer learning algorithms that enable policies trained in simulation to be deployed on real robots. By representing the skills obtained in simulation with a latent model, we achieve fast adaption of simulation-trained locomotion policies to a biped robot and a quadruped robot in a handful of trials (15-25). Our algorithms are generic and are more effective in overcoming large discrepancies between simulation and real robots than commonly used techniques such as domain randomization.

A DRL algorithm for safely transferring policies to novel environments. We propose a safe DRL algorithm for training policies that can robustly generalize to novel environments. By training a universal safe policy and an associated safety critic model, we can intelligently select between using the safe policy and the task policy during deployment. We demonstrate on a set of locomotion tasks that with our approach, the policy can be transferred to significantly different environments with at most two failure trials.

CHAPTER 2
RELATED WORK

In this Chapter, we present a summary of prior research work that is most relevant to our algorithms. Our work aims to create locomotion policies for physically-simulated characters and physical robots that can safely generalize to novel situations. We will first give a review of research in acquiring locomotion gaits for both simulated characters and real robots. We then discuss transfer learning algorithms that aims to transfer simulation-trained policies to real robots. Finally, we examine existing method in the field of safe deep reinforcement learning.

2.1 Locomotion Synthesis

Researchers in computer graphics and robotics have conducted extensive research in controlling simulated characters or real robots to walk or run. Among the earliest work in robotics that creates walking machines, Raibert and his colleagues built a series of legged robots and developed control algorithms that allow these robots to walk, jump, climb stairs, and perform gymnastic movements [17, 18, 19, 20, 21]. These robots and algorithms laid the foundations for modern legged robots such as Boston Dynamics Atlas and BigDog that keeps creating jaw-dropping movements. In computer graphics, Hodgins *et al.* presented a seminal work that demonstrated sophisticated human motions such as running, gymnastic vaulting and biking through physics-based simulation and control [22]. Since then, researchers in graphics have investigated a large variety of locomotion forms for different types of characters such as walking [23], flying [24], swimming [25], and bicycle stunts [26].

Due to the challenges in synthesizing locomotion gaits from scratch, existing algorithms usually require breaking the motion into more manageable parts or using motion

data to provide guidance to the synthesis algorithm. One approach is the use of finite state machines (FSMs) [27, 23, 28, 29, 30, 31, 32, 33, 34, 35, 36]. Yin *et al.* used FSMs to connect keyframes of the character poses, which is then combined with feedback rules to achieve balanced walking, skipping and running motion for humanoid characters [23]. de Lasa *et al.* presented a different application of FSMs, where they formulated the locomotion task as a Quadratic Programming (QP) problem and used an FSM model to switch between different objective terms to achieve walking motion [29]. FSMs have also been applied to construct hybrid dynamics systems for legged robots, where the state space of the robot is decomposed into multiple regions, each modeled by a continuous dynamics system. The resulting hybrid system is then used in trajectory optimization or policy optimization to generate locomotion gaits for robots [36, 37, 34, 38]. Although this class of techniques can successfully generate locomotion gaits, it is usually difficult to generalize them to more complex morphologies or arbitrary tasks.

An alternative approach to synthesize locomotion gaits is to incorporate motion data such as videos [39] or motion capture data [40, 41, 42, 43, 44, 45, 46]. By constraining motions to be similar to the existing motion data, we can obtain locomotion gaits that better resemble those seen in the real world. Recently, this line of work has also been extended to learning locomotion gaits for real quadruped robot [47]. Despite the high fidelity motion this approach can generate, the requirement of motion data can be limiting in applications that involves non animal-like characters or robots and in generalization to novel tasks.

Apart from designing finite state machines or using motion data, reward engineering has also been frequently applied to generate physically-based locomotion synthesis [48, 49, 50, 51, 52]. Al Borno *et al.* demonstrated a variety of humanoid motor skills by breaking a sequence of motion into shorter windows, and for each window a task-specific objective is optimized [48]. Mordatch *et al.* applied the Contact-Invariant-Optimization algorithm to generate full-body humanoid locomotion with muscle-based lower-body actuations [50]. Symmetry and periodicity of the motion was explicitly enforced to generate realistic loco-

motion.

The aforementioned methods can be understood as different ways of providing prior knowledge about the desired motion in order to achieve successful optimization of the gaits and ensuring the results look reasonable. Recent advancements in deep reinforcement learning (DRL) have shown the possibility of finding these complex locomotion gaits from simple and generic biomechanical principles without the use of explicit prior knowledge [53, 54, 55, 56]. For example, by combining TRPO with Generalized Advantage Estimation [57], Schulman *et al.* demonstrated learning of locomotion controllers for a 3D humanoid character. Despite the impressive feat of tackling the 3D biped locomotion problem from scratch, the resulting motion usually looks jerky and unnatural. In this thesis, we develop an algorithm that can synthesize believable locomotion controllers while refraining from using specific prior knowledge such as motion data or FSMs.

2.2 Sim-to-Real Transfer

Despite the tremendous progress made in DRL algorithms for acquiring complex motor skills in an automatic way, existing methods usually cannot be applied directly on real robots due to the large amount of experience required and safety concerns. A promising approach for mitigating the large amount of data required on the hardware while enjoying the autonomy of DRL algorithms is to train the control policies in computer simulated environments and transfer them to the real robots. However, a policy trained in simulation usually do not directly work on the real robot due to the presence of the Reality Gap [10].

Researchers have proposed a variety of techniques to overcome the reality gap [58, 14, 59, 60, 61, 62]. One important strategy is to improve the computer simulation to better match the real robot dynamics [58, 59, 62]. For example, Tan et al. [58] improved the actuator dynamics by identifying a nonlinear torque-current relation and demonstrated successful transfer of locomotion policies for a quadruped robot. However, improving the computer simulation alone is not sufficient to handle all the possible scenarios that the robot

might encounter in the real-world.

Another important technique for sim-to-real transfer is to train control policies that are robust to a range of simulated environments and sensor noises. Different techniques have been proposed to train robust policies, such as domain randomization [63, 60, 61, 64], adversarial perturbation [65], and ensemble models [66, 67]. Though training a policy with pure domain randomization may transfer to the real robot, it usually assumes that the training dynamics are not too far from the target dynamics. As shown in our experiments, domain randomization alone fails to transfer if the reality gap is large. In addition, without a mechanism to adjust the policy behavior, these policies cannot quickly adapt to cases where the reward function is changed.

In vision community, researchers have also investigated the problem of sim-to-real transfer to overcoming the discrepancies between rendered and real images. One of the most successful methods is domain adaptation [68, 69, 70]. It trains a generative model to transform the observations from the source domain to the target domain or both to a common intermediate domain. In this work, the main challenge is to adapt policies for dynamic changes, which is very different from visual changes.

To adapt to new reward functions or dynamics, it is necessary that the controller can modify its behavior according to the real-world experience. Existing works in this line of research can be roughly classified into two categories: model-free adaptation method and model-based adaptation method.

In model-free adaptation method, the control policy is directly adjusted according to experience from the target environment. One class of such method is the gradient-based meta learning approach [71, 72, 73, 74, 75], where the goal is to train policies that can be quickly adapted by gradient-based optimization methods during test time. Gradient-based meta learning methods have been demonstrated on adapting to novel reward function and are universal in theory [76]. However, it is in general less effective for adapting to novel dynamics. No-Reward Meta Learning (NoRML) [74] addressed this issue by meta-learning

an advantage function and an offset in addition to the policy parameters. NoRML has demonstrated effective adaptation to unseen dynamics in simulation. However, it has yet been demonstrated on real robots. More recently, Song *et al.* developed a sampling-based meta-learning algorithm based Hill-Climbing operator to adapt policy to new environments [75]. They demonstrated successful transfer from simulation to a real quadruped robot and adaptation to novel situations such as extra load within 50 trials.

In contrast to gradient-based method, latent space based adaptation method encodes the training experience into a latent representation [77, 12, 13, 78, 79]. The policy is then fine-tuned when a new environment is presented. Most methods in this class try to infer the latent input using observations from the target environment. For example, Yu et al. [12] conditioned the policy on the physics parameters of the robot, and trained a separate prediction model that estimates the physics parameters given the history of observations and actions. These methods can potentially adapt to changes in environments in an online fashion. However, when the dynamics changes significantly, the inference model may produce non-optimal latent inputs. As a result, most works have been demonstrated in simulated environments only.

Instead of training an inference model, researchers have also proposed methods that directly optimizes the latent input to the policy in the target environment [14, 13]. As the latent space that the policy is conditioned on is usually low dimensional, it is possible to use sampling-based optimization methods such as CMA-ES [80], or Bayesian Optimization [81] to find the best latent input that achieves the highest performance. Such methods have been successfully applied to learning locomotion policies for a biped robot [14]. Our method extends this line of research by matching the process of optimizing latent input during training and testing. We demonstrate that by doing this, we learn a better latent space that is suitable for fast adaptation.

Another related line of work is to define a space of robot behaviors, and then optimize on the real robot [82, 83]. For example, Cully et al. demonstrated fast adaptation on a

hexapod robot, by precomputeing a behavior-performance map and using Bayesian Optimization to search the map for the optimal controller when the robot is damaged [82]. Rai et al. also used Bayesian Optimization to optimize locomotion controllers for the ATRIAS Biped with less than 10 trials on the robot [83]. These approaches usually require designing a low-dimensional behavior space using domain knowledge. In contrast, our method applies meta learning, which leverages many different dynamic environments in training, to implicitly shape the lower-dimensional search space for the on-robot optimization.

Model-based adaptation method, on the other hand, adapts the dynamics model learned in source domain and extracts the control policy using methods such as model-predictive control (MPC) [84, 85, 86, 87, 88]. These methods have the advantage of being data efficient and can naturally adapt to changes in the environment online. However, performing inference of the optimal action in an MPC style is more computationally expensive, and the learned dynamics model usually uses the full state of the robot, which requires additional instruments such as a motion capture system.

2.3 Safe Reinforcement Learning

Advancements in robotics hardware and control algorithms have driven robots to be increasingly nimble, powerful, and intelligent. However, despite our ability to teach robots to perform human or animal-like skills such as walking, lifting objects, or even gymnastic movements, they are still limited in an ideal lab environment. One of the main gap robots need to overcome in order to be applied to real-world applications is safety.

The concept of safety for robots has taken many different forms in the literature. For example, prior work has related safety to the phenomena that in a stochastic environment, even the optimal policy can sometimes perform poorly [89, 90, 91]. In these work, they developed learning algorithms that not only maximizes the expected long-term reward of the policy, but also minimizes the variance of the rewards.

Another way to frame the safety of a robot is to relate it to the physical conditions of

the robot and the environment its in, e.g. if the robot breaks its end-effector, or damages objects nearby, it is considered unsafe. Our work falls into this category. The concept of safety in this framework can usually be defined as a constraint on the state of the robot. For example, a quadrotor should stay within states that do not collide with external objects. Thus it turns the motion synthesis problem into a constrained optimization problem. Some of the existing methods enforce these state constraints strictly [92, 93, 94, 95, 96, 97]. Given an initial safe but sub-optimal policy, Garcia *et al.* [92] and Berkenkamp *et al.* [95] develop exploration schemes that are provably safe around the current policy and gradually expand the region that the policy can operate within. They demonstrated successful training of policies for continuous control problems like inverted pendulum with few or none failure trials during the training. Ames *et al.* discussed the application of control barrier functions for finding safety-critical controllers that can keep the robot within an invariant set [97]. They provided a quadratic program formulation that minimally modifies an action provided by the nominal controller such that the filtered action is guaranteed to be safe and demonstrated the method on a few robotic control problems such as biped robot walking on stepping stones. Despite being able to provide guarantees on the safety, these methods usually assume the availability of system dynamics or the ability to gradually build the system dynamics by leveraging an initial controller that is safe.

Alternatively, researchers have also investigated methods that converges to a policy that satisfies the constraints in expectation, but do not provides guarantees during the learning [98, 99, 100, 101, 102, 103]. For example, Achiam *et al.* introduced a general DRL algorithm named Contrained Policy Optimization (CPO) that solves a constrained MDP problem [98]. They demonstrated CPO on simulated continuous contorl tasks where the agent needs to complete the tasks while satisfying certain state constraints. Berkenkamp *et al.* used Gaussian Processes (GPs) to model the performance and safety of the policy [102]. Using the trained GPs, they can adaptively evaluate new policies to improve the performance, while ensuring high probability of safety. Although these methods do not

require building the dynamics of the system, they do not provide guarantees on the safety, making them not applicable to safety-critical problems.

In addition to enforcing general safety constraints, researchers have also investigated more task-dependent safety issues. For example, for humanoid robots, researchers have devised specialized algorithms to reduce the damage they receives during falling [104, 105, 106, 107]. Kumar *et al.* used deep reinforcement learning algorithm to find a controller that minimizes the impact from falling for a humanoid robot [107]. Though effective in handling the falling scenario, these methods may not generalize to other types of failure modes for robots.

CHAPTER 3

BACKGROUND

3.1 Markov Decision Process

We formulate the locomotion learning problem as a Markov Decision Process (MDP), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, p_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ is the transition function, p_0 is the initial state distribution and γ is the discount factor. In practice, we usually only have access to an observation of the robot that contains partial information of the robot's state. In this case, we will have a Partially-Observable Markov Decision Process (POMDP) and the policy would become $\pi : \mathcal{O} \mapsto \mathcal{A}$, where $\mathcal{O}$ is the observation space. In the context of sim-to-real transfer, we can define the learning problem in simulation as an MDP $\mathcal{M}^s$ and the real robot problem as another MDP $\mathcal{M}^r$. We can further parameterize the simulation dynamics as $\mathbf{s}_{t+1} = \mathcal{P}_{\boldsymbol{\mu}}(\mathbf{s}_t, \mathbf{a}_t)$, where $\boldsymbol{\mu}$ is a vector of physical parameters defining the dynamic model (e.g. friction coefficient).

The goal of reinforcement learning is to find a control policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maximizes the expected accumulated reward:

$$J_{\mathcal{M}}(\theta) = \mathbb{E}_{\tau=(\mathbf{s}_0,\mathbf{a}_0,\ldots,\mathbf{s}_T)} \sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t), \tag{3.1}$$

where $\mathbf{s}_0 \sim p_0$, $\mathbf{a}_t \sim \pi(\mathbf{s}_t)$, $\mathbf{s}_{t+1} = \mathcal{P}(\mathbf{s}_t, \mathbf{a}_t)$ and θ is the parameters of the policy, usually represented as a neural network. A more compact way to represent this optimization is using the Q function:

$$\pi_{\theta^*} = \arg\max_{\theta} \mathbb{E}_{\mathbf{s}\sim p_0}[Q^{\pi}(\mathbf{s}, \mathbf{a})], \tag{3.2}$$

where the Q function of a policy, $Q^\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, is defined as the expected long-term reward of taking an action $\mathbf{a}$ and then follow the policy π_θ from some input state $\mathbf{s}_t$:

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\mathbf{s}_t, \mathbf{a}_t), \mathbf{a}_{t+1} \sim \pi(\mathbf{s}_{t+1}), \ldots}[\sum_{i=0}^{\infty} \gamma^i r(\mathbf{s}_{t+i}, \mathbf{a}_{t+i})]. \tag{3.3}$$

For a state $\mathbf{s}$, if we take an expectation over the actions for the Q function, we obtain the value function for this state:

$$V^\pi(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{s}_t)} Q^\pi(\mathbf{s}_t, \mathbf{a}). \tag{3.4}$$

Taking the different between Q function and value function, we get the advantage function $A^\pi(\mathbf{s}, \mathbf{a}) = Q^\pi(\mathbf{s}, \mathbf{a}) - V^\pi(\mathbf{s})$, which indicates the improvement we can achieve in expectation if we take action $\mathbf{a}$ instead of following the policy π.

3.2 Policy Search Algorithms

Policy search methods have demonstrated success in solving high-dimensional, continuous MDPs. In this thesis, we use a total of three policy search algorithms for different projects: Trust Region Poicy Optimization (TRPO) [55], Proximal Policy Optimization (PPO) [56], and Augmented Random Search (ARS) [108]. In the following sections, we provide a brief review of the three algorithms. We refer readers to the corresponding original paper for more details.

3.2.1 Trust-Region Policy Optimization

Trust-Region Policy Optimization (TRPO) [55] starts by expressing the expected return of the updated policy $\tilde{\pi}$ using the advantage function of the current policy π:

$$J(\tilde{\pi}) = J(\pi) + \mathbb{E}_{\mathbf{s}_0, \mathbf{a}_0, \cdots \sim \tilde{\pi}}[\sum_{t=0}^{T} \gamma^t A^\pi(\mathbf{s}_t, \mathbf{a}_t)]. \tag{3.5}$$

By defining the unnormalized state visitation frequency:

$$\rho_\pi(\mathbf{s}) = P(\mathbf{s}_0 = \mathbf{s}) + \gamma P(\mathbf{s}_1 = \mathbf{s}) + \gamma^2 P(\mathbf{s}_2 = \mathbf{s}) + \dots,$$

we can rewrite Equation 3.5 as:

$$J(\tilde{\pi}) = J(\pi) + \sum_{\mathbf{s}} \rho_{\tilde{\pi}}(\mathbf{s}) \sum_{\mathbf{a}\sim\tilde{\pi}(\mathbf{s})} A^\pi(\mathbf{s}, \mathbf{a}). \tag{3.6}$$

Directly working with Equation 3.6 is difficult due to that the state visitation frequency $\rho_{\tilde{\pi}}$ depends on the updated policy $\tilde{\pi}$. To make the problem more manageable, TRPO use a local approximation of Equation 3.6 by assuming the state visitation frequency does not change when updating the policy:

$$L(\tilde{\pi}) = J(\pi) + \sum_{\mathbf{s}} \rho_{\tilde{\pi}}(\mathbf{s}) \sum_{\mathbf{a}\sim\pi(\mathbf{s})} A^\pi(\mathbf{s}, \mathbf{a}). \tag{3.7}$$

This approximation holds when the difference between the old policy π and the updated policy $\tilde{\pi}$ is small. This implies that if we take a sufficiently small step in the policy space that maximizes Equation 3.7, we can obtain an updated policy that achieves better return than the old policy. In order to do this, TRPO formulates the following constrained optimization problem, note that we assume the policy π is parameterized by θ and we use θ to denote π_θ to reduce the cumbersome:

$$\begin{aligned} &\max_\theta L(\theta, \theta_{old}) \\ &\text{subject to } \bar{D}_{KL}(\pi_{\theta_{old}} | \pi_\theta) \leq \sigma, \end{aligned} \tag{3.8}$$

where $\bar{D}_{KL}$ is the mean KL divergence of two distributions and σ is the maximum step size we are allowed to take in the policy space, measured by KL divergence. In the case where we use a sampling-based approach to estimate the objective function and the constraints,

we can reach the optimization problem below:

$$\max_{\theta} \mathbb{E}_{\mathbf{s}\sim\rho_{\theta_{old}},\mathbf{a}\sim\pi_{\theta}}[\frac{\pi_{\theta}}{\pi_{\theta_{old}}}A^{\theta_{old}}(\mathbf{s},\mathbf{a})] \tag{3.9}$$
$$\text{subject to } \mathbb{E}_{\mathbf{s}\sim\rho_{\theta_{old}}}[KL(\pi_{\theta_{old}}|\pi_{\theta})] \leq \sigma,$$

where $\frac{\pi_{\theta}}{\pi_{\theta_{old}}}$ is the importance sampling ratio that allows us to compute the expectation of actions over the updated policy π_{θ} while using samples from the old policy $\pi_{\theta_{old}}$.

At each iteration, TRPO computes a search direction, $\mathbf{d} = H^{-1}\mathbf{g}$, where H is the Fisher information matrix of the constraint that approximates the KL-divergence and $\mathbf{g}$ is the first-order derivative of the objective function (Equation 3.9). To ensure that the KL-divergence constraint is satisfied, the search direction is first scaled to the boundary of approximated KL-divergence by multiplying the scaling factor, $\sqrt{\frac{2\sigma}{\mathbf{g}^T H^{-1}\mathbf{g}}}$. An iterative line search is then performed from the scaled $\mathbf{d}$ to guarantee that the KL-divergence constraint is satisfied, while maximizing the objective function.

3.2.2 Proximal Policy Optimization

Similar to TRPO, Proximal Policy Optimization (PPO) [56] is also based on the idea of maximizing the approximated policy return (Equation 3.7) while making sure the difference between the new and old policy is small. However, different from TRPO which formulates the problem as a constrained optimization problem, PPO enforces the size of the update in the policy through a clipping mechanism. Specifically, if we define the importance sampling ratio as $\eta = \frac{\pi_{\theta}}{\pi_{\theta_{old}}}$, PPO solves the following optimization problem:

$$\max_{\theta} \mathbb{E}_{\mathbf{s}\sim\rho_{\theta_{old}},\mathbf{a}\sim\pi_{\theta}}[\min(\eta A^{\theta_{old}}(\mathbf{s},\mathbf{a}), \text{clip}(\eta, 1-\epsilon, 1+\epsilon)A^{\theta_{old}}(\mathbf{s},\mathbf{a}))], \tag{3.10}$$

where ϵ controls the step size of the policy update and is typically chosen to be 0.2. PPO has also been demonstrated to be effective in high dimensional continuous control problems and it is in general more efficient than TRPO due to the simpler optimization formulation.

3.2.3 Augmented Random Search

Augmented Random Search (ARS) [108] presents a different strategy to optimize a control policy. Different from policy gradient-based methods such as TRPO and PPO, ARS uses a sampling-based approach to estimate the gradient of the policy return with respect to the policy parameters θ.

The core idea of ARS is to sample N perturbations in the policy parameter space following a multi-variate normal distribution $\mathcal{N}(0, \nu)$. For each perturbation ξ_i, we evaluate the perturbed policy performance in both directions: $J(\theta + \xi_i)$ and $J(\theta - \xi_i)$. We can obtain the estimated gradient in the direction of ξ_i using tow sided finite-difference: $\mathbf{g}_i = \frac{J(\theta+\xi_i)-J(\theta-\xi_i)}{\nu}\xi_i$. The final update to the policy can then be computed by taking the average over $\mathbf{g}_i$: $\mathbf{g} = \frac{\sigma_{i=0}^{N}\mathbf{g}_i}{N}$. However, this vanilla sampling-based algorithm alone is not sufficient to achieve good performance for optimizing the control policy. To obtain stable and effective learning, ARS makes multiple modifications to the vanilla version of the algorithm.

First, they found that normalizing the observation $\mathbf{o}$ input to the policy is important. Specifically, they maintain a running mean and co-variance matrix of the observations seen so far: $\mu_{\mathbf{o}}$, $\Sigma_{\mathbf{o}}$, and apply it to the observations before feeding to the policy: $\tilde{\mathbf{o}} = \text{diag}(\Sigma_{\mathbf{o}})^{-\frac{1}{2}}(\mathbf{o} - \mu_{\mathbf{o}})$. Second, they observe that as the policy improves, the magnitude of the evaluated gradient $\mathbf{g}_i = \frac{J(\theta+\xi_i)-J(\theta-\xi_i)}{\nu}\xi_i$ will also increase, leading to unstable learning. To address this, they maintain the standard deviation σ_R of the evaluated policy returns for the current iteration and use it to scale the update steps. Finally, they discard the worst performing perturbations in a batch of samples, which was found to improve the learning performance.

Because ARS samples in the policy space to estimate the gradient, it is in general less sample efficient than policy gradient methods that samples in the action space when optimizing a neural network policy. However, ARS can better handle sparse rewards and can better utilize large-scale parallel computing.

CHAPTER 4

LEGGED LOCOMOTION WITH DEEP REINFORCEMENT LEARNING

4.1 Motivation

Animals in the nature have develop a variety of skills to gracefully move in their natural habitats, such as flying, swimming, and running. Reproducing these elegant locomotion skills on animated characters is a fascinating challenge for graphics researchers and animators. Knowledge from biomechanics, physics, robotics, and animation give us ideas for how to coordinate the virtual muscles of a character's body to move it forward while maintaining balance and style. Whether physical or digital, the creators of these characters apply physics principles, borrow ideas from domain experts, use motion data, and undertake arduous trial-and-error to craft lifelike movements that mimic real-world animals and humans. While these characters can be engineered to exhibit locomotion behaviors, a more intriguing question is whether the characters can learn locomotion behaviors on their own, the way a human toddler can.

The recent disruptive development in Deep Reinforcement Learning (DRL) suggests that the answer is yes. Researchers indeed showed that artificial agents can learn some form of locomotion using advanced policy learning methods with a large amount of computation. Even though such agents are able to move from point A to point B without falling, the resulting motion usually exhibits jerky, high-frequency movements. The motion artifacts can be mitigated by introducing motion examples or special objectives in the reward function, but these remedies are somewhat unsatisfying as they sacrifice generality of locomotion principles and return partway to heavily engineered solutions.

In this Chapter, we introduce a DRL algorithm that takes a minimalist approach to the problem of learning locomotion. Our hypothesis is that natural locomotion will emerge

from simple and well-known principles in biomechanics, without the need of motion examples or morphology-specific considerations. Our approach addresses two common problems in the motions produced by the existing methods. First, we observe that the motions appear much more energetic than biological systems. Second, an agent with perfectly symmetrical morphology often produces visibly asymmetrical motion, contradicting the observation in biomechanics literature that gaits are statistically symmetrical [109].

Our algorithm consists of two main components: a curriculum learning algorithm for achieving successful locomotion policies that aggressively penalize energy consumption, and a mirror symmetry constraint that encourages symmetry in the motion. Our curriculum provides modulated physical assistance appropriate to the current skill level of the learner, ensuring continuous progress toward successful locomotion with low energy consumption. Our algorithm automatically computes the assistive forces to help the character with lateral balance and forward movement and gradually relaxes the assistance, so that eventually the character learns to move without help. Our solution for encouraging symmetric locomotion departs from the conventional metrics that measure the symmetry of the states. Instead, we measure the *symmetry of actions* produced by the policy. Our formulation is simple, generic, and is flexible in supporting asymmetric motions when desired.

Our evaluation shows that the agent can indeed learn locomotion that exhibits symmetry and speed-appropriate gait patterns and consumes relatively low-energy, without the need of motion examples. Our method can be applied to a variety of morphologies, such as bipeds, quadrupeds, or hexapods. We test our method against three baselines: learning without the mirror symmetry loss, learning without the curriculum, and learning without either component. The comparisons show that, without the curriculum learning, the trained policies fail to move forward or/and maintain balance. On the other hand, without the mirror symmetry loss, the learning process takes significantly more trials and results in asymmetric locomotion.

4.2 Learning Locomotion Policy

In this work, we define a state as $\mathbf{s} = [\mathbf{q}, \dot{\mathbf{q}}, \mathbf{c}, \hat{v}]$, where $\mathbf{q}$ and $\dot{\mathbf{q}}$ are the joint positions and joint velocities. $\mathbf{c}$ is a binary vector with the size equal to the number of end-effectors, indicating the contact state of the end-effectors (1 in contact with the ground and 0 otherwise). $\hat{v}$ is the target velocity of the center of mass in the forward direction. The action $\mathbf{a}$ is simply the joint torques generated by the actuators of the character.

Designing a reward function is one of the most important tasks in solving a MDP. In this work, we use a generic reward function for locomotion similar to those used in RL benchmarks [110] [111] [112]. It consists of three objectives: move forward, balance, and use minimal actuation.

$$r(\mathbf{s}, \mathbf{a}) = w_v E_v(\mathbf{s}) + E_u(\mathbf{s}) + w_l E_l(\mathbf{s}) + E_a + w_e E_e(\mathbf{a}). \tag{4.1}$$

The first term of the reward function, $E_v = -|\bar{v}(\mathbf{s}) - \hat{v}|$, encourages the character to move at the desired velocity $\hat{v}$. $\bar{v}(\mathbf{s})$ denotes the average velocity in the most recent 2 seconds. The next three terms are designed to maintain balance. $E_u = -(w_{u_x}|\phi_x(\mathbf{s})| + w_{u_y}|\phi_y(\mathbf{s})| + w_{u_z}|\phi_z(\mathbf{s})|)$ rewards the character for maintaining its torso or head upright, where $\phi(\mathbf{s})$ denotes the orientation of the torso or head. $E_l = -|c_z(\mathbf{s})|$ penalizes deviation from the forward direction, where $c_z(\mathbf{s})$ computes the center of mass (COM) of the character in the frontal axis. E_a is the alive bonus which rewards the character for not being terminated at the current moment. A rollout is terminated when the character fails to keep its COM elevated along the forward direction, or to keep its global orientation upright. Finally, $E_e = -\|\mathbf{a}\|$ penalizes excessive joint torques, ensuring minimal use of energy. Details on the hyper-parameters related to the reward function and the termination conditions are discussed in Section 4.5.

4.3 Locomotion Curriculum Learning

Learning locomotion directly from the principles of minimal energy and gait symmetry is difficult without additional guidance from motion examples or reward function shaping. Indeed, a successful locomotion policy must learn a variety of tasks often with contradictory goals, such as maintaining balance while propelling the body forward, or accelerating the body while conserving energy. One approach to learning such a complex motor skill is to design a curriculum that exposes the learner to a series of tasks with increasing difficulty, eventually leading to the original task.

Our locomotion curriculum learning is inspired by physical learning aids that provide external forces to simplify the motor tasks, such as exoskeletons for gait rehabilitation or training wheels for riding a bicycle. These learning aids create a curriculum to ease the learning process and will be removed when the learner is sufficiently skilled at the original task. To formalize this idea, we view the curriculum as a continuous Euclidean space parameterized by curriculum variables $\mathbf{x} \in \mathbb{R}^n$. The learning begins with the simplest lesson $\mathbf{x}_0$ for the beginner learner, gradually increasing the difficulty toward the original task, which is represented as the origin of the curriculum space (i.e. $\mathbf{x} = \mathbf{0}$). With this notion of a continuous curriculum space, we can then develop a continuous learning method by finding the optimal path from $\mathbf{x}_0$ to the origin in the curriculum space.

Similar to the standard policy gradient method, at each learning iteration, we generate rollouts from the current policy, use the rollout to estimate the gradients of the objective function of policy optimization, and update the policy parameters θ based on the gradient. With curriculum learning, we introduce a virtual assistant to provide assistive forces to the learner during rollout generation. The virtual assistant is updated at each learning iteration such that it provides assistive forces appropriate to the current skill level of the learner.

Two questions remain in our locomotion curriculum learning algorithm. First, what is the most compact set of parameters for the virtual assistant such that locomotion skills

can be effectively learned through curriculum? Second, what is the appropriate curriculum schedule, i.e. how much assistive force should we give to the learner at each moment of learning?

4.3.1 Virtual Assistant

Our virtual assistant provides assistive forces to simplify the two main tasks of locomotion: moving forward and maintaining lateral balance. The lateral balancing force is applied along the frontal axis (left-right) of the learner, preventing it from falling sideway. The propelling force is applied along the sagittal axis, pushing the learner forward to reach the desired velocity. With these two assistive forces, the learner can focus on learning to balance in the sagittal plane as well as keeping the energy consumption low.

Both lateral balancing force and propelling force are produced by a virtual proportional-derivative (PD) controller placed at the pelvis of the learner, as if an invisible spring is attached to the learner to provide support during locomotion. Specifically, the PD controller controls the lateral position and the forward velocity of the learner. The target lateral position is set to 0 for maintaining lateral balance while the target forward velocity is set to the desired velocity $\bar{v}$ for assisting the learner moving forward.

Different levels of assistance from the virtual assistant create different lesson for the learner. We use the stiffness coefficient k_p and the damping coefficient k_d to modulate the strength of the balancing and propelling forces respectively. As such, our curriculum space is parameterized by $\mathbf{x} = (k_p, k_d)$. Any path from $\mathbf{x}_0$ to $(0, 0)$ constitutes a curriculum for the learner.

Our implementation of the virtual assistant is based on the stable proportional-derivative (SPD) controller proposed by Tan *et al.* [113]. The SPD controller provides a few advantages. First, it does not require any pre-training and can be applied to any character morphology with little tuning. In addition, it provides a smooth assistance in the state space, which facilitates learning. Finally, it is unconditionally stable, allowing us to use large

controller gains without introducing instability.

4.3.2 Curriculum Scheduling

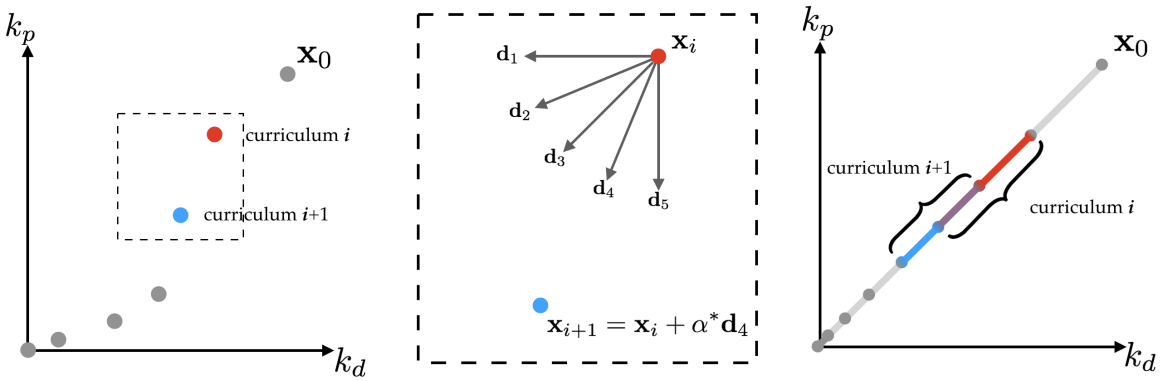


Figure 4.1: (a) The learner-centered curriculum determines the lessons adaptively based on the current skill level of the agent, resulting in a piece-wise linear path from $\mathbf{x}_0$ to the origin. (b) In each learning iteration, the learner-centered curriculum finds the next point in the curriculum space using a simple search algorithm that conducts 1D line-searches along five direction. The goal is to find the largest step size, α^*, such that the current policy can still reach 60% of the original return $\bar{R}$. (c) Environment-centered curriculum follows a series of predefined lessons along a linear path from $\mathbf{x}_0$ to the origin. It introduces the learner to a range of lessons in one curriculum learning iteration, resulting in a set of co-linear, overlapping line segments.

The goal of curriculum scheduling is to systematically and gradually reduce the assistance from the initial lesson $\mathbf{x}_0$ and ultimately achieve the final lesson in which the assistive force is completely removed. Designing such a schedule is challenging because an aggressive curriculum that reduces the assistive forces too quickly can fail the learning objectives while a conservative curriculum can lead to inefficient learning.

We propose two approaches to the problem of curriculum scheduling (Figure 4.1): Learner-centered curriculum and Environment-centered curriculum. The learner-centered curriculum allows the learner to decide the next lesson in the curriculum space, resulting in a piece-wise linear path from $\mathbf{x}_0$ to the origin. The environment-centered curriculum, on the other hand, follows a series of predefined lessons. However, instead of focusing on

one lesson at a time, it exposes the learner to a range of lessons in one curriculum learning iteration, resulting in a set of co-linear, overlapping line segments from $\mathbf{x}_0$ to the origin of the curriculum space.

Learner-centered curriculum

The learner-centered curriculum determines the lessons adaptively based on the current skill level of the agent (Algorithm 1). We assume that the initial lesson $\mathbf{x}_0$ is sufficiently simple such that the standard policy learning algorithm can produce a successful policy π, which generates rollouts $\mathcal{B}$ with average return, $\bar{R}$. We then update the lesson to make it more challenging (Algorithm 2) and proceed to the main learning loop. At each curriculum learning iteration, we first update π by running one iteration of the standard policy learning algorithm. If the average of the rollouts from the updated policy can reach $h\%$ of the original return $\bar{R}$ ($h = 80$), we update the lesson again using Algorithm 2. The curriculum learning loop terminates when the magnitude of $\mathbf{x}$ is smaller than ϵ ($\epsilon = 5$). We run a final policy learning without any assistance, i.e. $\mathbf{x} = (0, 0)$. At this point, the policy learns this final lesson very quickly.

Given the current lesson $\mathbf{x}_i$, the goal of Algorithm 2 is to find the next point in the curriculum space that is the closest to the origin while the current policy can still retain some level of proficiency. Since it is only a two-dimensional optimization problem and the solution $\mathbf{x}_{i+1}$ lies in $\|\mathbf{x}_{i+1}\| - \|\mathbf{x}_i\| < 0$ and is strictly positive component-wise, we implement a simple search algorithm that conducts 1D line-searches along five directions from $\mathbf{x}_i$: (-1, 0), (-1, -0.5), (-1, -1), (-0.5, -1), (0, -1). For each direction $\mathbf{d}$, the line-search will return the largest step size, α, such that the current policy can still reach $l\%$ of the original return $\bar{R}$ ($l = 60$) under the assistance $\mathbf{x}_i + \alpha\mathbf{d}$. Among five line-search results, we choose the largest step size, α^* along the direction $\mathbf{d}^*$ to create the next lesson: $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha^*\mathbf{d}^*$.

Environment-centered curriculum

Instead of searching for the next lesson, the environment-centered curriculum updates the lessons along a predefined linear path from $\mathbf{x}_0$ to the origin (Algorithm 3). In addition, the learner is trained with a range of lessons $[\mathbf{x}_{begin}, \mathbf{x}_{end}]$ in each curriculum learning iteration. Specifically, the learner will be exposed to an environment in which the virtual assistant starts with $\mathbf{x}_{begin}$ and reduces its strength gradually to $\mathbf{x}_{end}$ at the end of each rollout horizon. The formula of strength reduction from $\mathbf{x}_{begin}$ to $\mathbf{x}_{end}$ can be designed in several ways. We use a simple step function to drop the strength of the virtual assistant by $k\%$ every p seconds ($k = 25$ and $p = 3$). Each step in the step function can be considered a learning milestone. Training with a range of milestones in a single rollout prevents the policy from overfitting to a particular virtual assistant, leading to more efficient learning.

In each learning iteration, we first run the standard policy learning for one iteration, with the environment programmed to present the current range of lessons $[\mathbf{x}_{begin}, \mathbf{x}_{end}]$. After the policy is updated, we evaluate the performance of the policy using two conditions. If the policy meets both conditions, we update the range of lessons to $k\% \cdot [\mathbf{x}_{begin}, \mathbf{x}_{end}]$. Note that the updated $\mathbf{x}_{begin}$ is equivalent to the second milestone of the previous learning iteration, resulting in some overlapping lessons in two consecutive curriculum learning iterations (See Figure 4.1c). The overlapping lessons are an important aspect of the environment-centered curriculum learning because they allow the character to bootstrap its current skill when learning a new set of predefined lessons. Similar to the learner-centered curriculum, the curriculum learning loop terminates when the magnitude of $\mathbf{x}_{begin}$ is smaller than ϵ ($\epsilon = 5$), and we run a final policy learning without any assistance, i.e. $\mathbf{x}_{begin} = (0, 0)$ and $\mathbf{x}_{end} = (0, 0)$.

The first condition for assessing the progress of learning checks whether the learner is able to reach the second milestone in each rollout. That is, the agent must stay balance for at least $2p$ seconds. Using this condition alone, however, might result in a policy that simply learns to stand still or move minimally in balance. Therefore, we use another condition that

Algorithm 1 Learner-Centered Curriculum Learning

$\mathbf{x} = \mathbf{x}_0$
$[\pi, \mathcal{B}] \leftarrow$ PolicyLearning($\mathbf{x}$)
$\bar{R} \leftarrow$ AvgReturn($\mathcal{B}$)
$\mathbf{x} \leftarrow$ UpdateLesson($\mathbf{x}$, $\bar{R}$, π)
while $||\mathbf{x}|| \geq \epsilon$ **do**
    $[\pi, \mathcal{B}] \leftarrow$ OneIterPolicyLearning($\mathbf{x}$)
    $R \leftarrow$ AvgReturn($\mathcal{B}$)
    if $R \geq h\% \cdot \bar{R}$ **then**
        $\mathbf{x} \leftarrow$ UpdateLesson($\mathbf{x}$, $\bar{R}$, π)
$[\pi, \mathcal{B}] \leftarrow$ PolicyLearning($(0, 0)$)
return π

Algorithm 2 Update Lesson

input: $\mathbf{x}, \bar{R}, \pi$
$\mathcal{D} = [(-1, 0), (-1, -0.5), (-1, -1), (-0.5, -1), (0, -1)]$
$\mathbf{x}_{min} = (\infty, \infty)$
for each $\mathbf{d} \in \mathcal{D}$ **do**
    $\alpha^* = \text{argmax}_\alpha \ \alpha$
        s.t. EvalReturn($\mathbf{x} + \alpha\mathbf{d}, \pi$) $> l\% \cdot \bar{R}$
    if $||\mathbf{x} + \alpha^*\mathbf{d}|| < ||\mathbf{x}_{min}||$ **then**
        $\mathbf{x}_{min} = \mathbf{x} + \alpha^*\mathbf{d}$
return $\mathbf{x}_{min}$

requires the average return of the policy to reach a pre-determined threshold, $\bar{R}$, which is $g\%$ of the return from the initial policy trained with full assistance ($g = 70$).

4.4 Mirror Symmetry Loss

Symmetry is another important characteristic of a healthy gait. Assessing gait symmetry usually requires at least an observation of a full gait cycle. This requirement poses a challenge to policy learning because the reward cannot be calculated before the end of the gait cycle, leading to a delayed reward function. We propose a new way to encourage gait symmetry by measuring the symmetry of *actions* instead of *states*, avoiding the potential issue of delayed reward.

Imaging a person who is standing in front of a floor mirror with her left hand behind her back. If she uses the right hand to reach for her hat, what we see in the mirror is a person

Algorithm 3 Environment-Centered Curriculum Learning

$\mathbf{x}_{begin} = \mathbf{x}_0$
$\mathbf{x}_{end} = (k^2)\% \cdot \mathbf{x}_{begin}$
$[\pi, \mathcal{B}] \leftarrow$ PolicyLearning($\mathbf{x}_{begin}, \mathbf{x}_{end}$)
$\bar{R} \leftarrow g\%\cdot$ AvgReturn($\mathcal{B}$)
while $||\mathbf{x}_{begin}|| \geq \epsilon$ **do**
    $[\pi, \mathcal{B}] \leftarrow$ OneIterPolicyLearning($\mathbf{x}_{begin}, \mathbf{x}_{end}$)
    if BalanceTest($\mathcal{B}$) and AvgReturn($\mathcal{B}$) $> \bar{R}$ **then**
        $\mathbf{x}_{begin} = k\% \cdot \mathbf{x}_{begin}$
        $\mathbf{x}_{end} = k\% \cdot \mathbf{x}_{end}$
$[\pi, \mathcal{B}] \leftarrow$ PolicyLearning($(0, 0), (0, 0)$)
return π

with her right hand behind her back reaching for a hat using her left hand. Indeed, if the character has a symmetric morphology, the action it takes in some pose during locomotion should be the mirrored version of the action taken when the character is in the mirrored pose. This property can be expressed as:

$$\pi_\theta(\mathbf{s}) = \Psi_a(\pi_\theta(\Psi_o(\mathbf{s}))), \tag{4.2}$$

where $\Psi_a(\cdot)$ and $\Psi_o(\cdot)$ maps actions and states to their mirrored versions respectively. We overload the notation π_θ to represent the mean action of the stochastic policy. Enforcing Equation 4.2 as a hard constraint is difficult for standard policy gradient algorithms, but we can formulate a soft constraint and include it in the objective function for policy optimization:

$$L_{sym}(\theta) = \sum_{i=0}^{B} ||\pi_\theta(\mathbf{s}_i) - \Psi_a(\pi_\theta(\Psi_o(\mathbf{s}_i)))||^2, \tag{4.3}$$

where B is the number of simulation samples per iteration. We use $20,000$ samples in all of our examples. Since Equation 4.3 is differentiable with respect to the policy parameters θ, it can be combined with the standard reinforcement learning objective and optimized using any gradient-based RL algorithm.

Incorporating the mirror symmetry loss, the final optimization problem for learning the

locomotion policy can be defined as:

$$\pi_{\theta^*} = \underset{\theta}{\operatorname{argmin}} \quad L_{PPO}(\theta) + wL_{sym}(\theta), \tag{4.4}$$

where w is the weight to balance the importance of the gait symmetry and the expected return of the policy ($w = 4$). Note that the mirror symmetry loss is included in the objective function of the policy optimization, rather than in the reward function of the MDP. This is because L_{sym} explicitly depends on the policy parameters θ, thus adding L_{sym} in the reward function would break the assumption in the Policy Gradient Theorem [114]. That is, changing θ should change the probability of a rollout, not its return. If we included L_{sym} in the reward function, it would change the return of the rollout when θ is changed. Instead, we include L_{sym} in the objective function and calculate its gradient separately from that of the L_{PPO}, which depends on the Policy Gradient Theorem.

Alternatively, one can also enforce symmetry as a hard constraint in the neural network architecture. Abdolhosseini *et al.* proposed one possible way to construct such model with symmetry enforcement [115], here we describe a simpler architecture to enforce symmetry in the neural network. We first define a neural network module $f_\theta : \mathcal{S} \mapsto \mathcal{A}$. We can then define the control policy as: $\pi_\theta(\mathbf{o}) = 0.5 * (f(\mathbf{o}) + \Phi_a(f(\Phi_o(\mathbf{o}))))$. It can be easily verified by plugging in that the resulting policy satisfies the symmetry constraint. Enforcing the symmetry constraint in the policy architecture allows it to be more flexible and applied in more algorithms.

4.5 Results

We evaluate our method on four characters with different morphologies and degrees of freedom. The input to the control policy includes $\mathbf{s} = [\mathbf{q}, \dot{\mathbf{q}}, \mathbf{c}, \hat{v}]$ and the output is the torque generated at each actuated joint, as described in Section 4.2. Because the character is expected to start at zero velocity and accelerate to the target velocity, the policy needs to

be trained with a range of velocities from zero to the target. We include $\hat{v}$ in the input of the policy to modulate the initial acceleration; $\hat{v}$ is set to zero at the beginning of the rollout and increases linearly for the first $0.5|\hat{v}|$ seconds, encouraging the character to accelerate at $2m/s^2$. The parameters of the reward functions used in all the experiments are listed in Table 4.1. We demonstrate the environment-centered curriculum learning for all the examples and selectively use the learner-centered curriculum learning for comparison. The resulting motions can be seen in the supplementary video.

We set the starting point of the curriculum $\mathbf{x}_0$ to $(k_p, k_d) = (2000, 2000)$ in all examples. Note that k_p and k_d are the proportional gains and damping gains in two independent SPD controllers that provide balancing and propelling forces respectively. The damping gain used to compute the balancing force is $0.1k_p$ and the proportional gain used to compute the propelling force is 0.

We use Pydart [116], a python binding of the DART library [5] to perform multi-body simulation. We simulate the characters at 500 Hz, and query the control policy every 15 simulation steps, yielding a control frequency of 33 Hz. We use the PPO algorithm implemented in the OpenAI Baselines library [117] for training the control policies. The control policies used in all examples are represented by feed-forward neural networks with three fully-connected hidden layers, and each hidden layer consists of 64 units. We fix the sample number to be $20,000$ steps per iteration for all examples. The number of iteration required to obtain a successful locomotion controller depends on the complexity of the task, ranging from 500 to 1500, yielding a total sample number between 10 and 30 millions. We perform all training using 8 parallel threads on an Amazon EC2 node with 16 virtual cores and 32G memory. Each training iteration takes $25 - 45$s depending on the degrees of freedoms of the character model, leading to a total training time between 4 and 15 hours.

Table 4.1: Task and reward parameters

Character	$\hat{v}$	w_v	w_{u_x}	w_{u_y}	w_{u_z}	w_l	E_a	w_e
Simplified Biped	0 to $1m/s$	3	1	1	1	3	4	0.4
Simplified Biped	0 to $5m/s$	3	1	1	1	3	7	0.3
Quadruped	0 to $2m/s$	4	0.5	0.5	1	3	4	0.2
Quadruped	0 to $7m/s$	4	0.5	0.5	1	3	11	0.35
Hexapod	0 to $2m/s$	3	1	1	1	3	4	0.2
Hexapod	0 to $4m/s$	3	1	1	1	3	7	0.2
Humanoid	0 to $1.5m/s$	3	1	1.5	1	3	6	0.3
Humanoid	0 to $5m/s$	3	1	1.5	1	3	9	0.15
Humanoid	0 to $-1.5m/s$	3	1	1.5	1	3	6	0.3

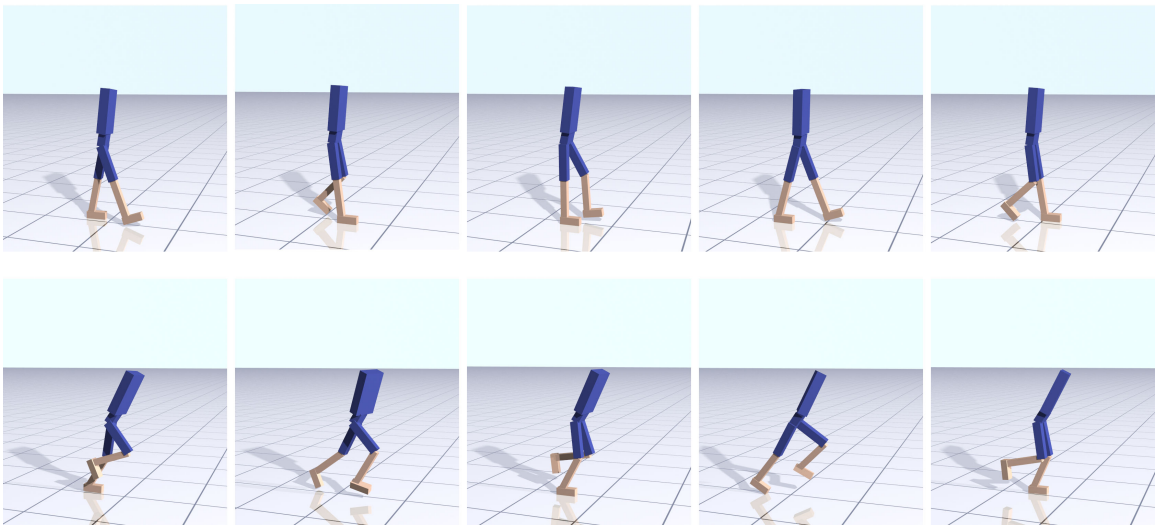


Figure 4.2: Simplified biped walking (top) and running (bottom). Results are trained using environment-centered curriculum learning and mirror symmetry loss.

4.5.1 Locomotion of Different Morphologies

Simplified biped Bipedal locomotion has been extensively studied in the literature, and it is a familiar form of locomotion to everyone. Thus we start with training a simplified biped character to perform walking and running. The character has 9 links and 21 DOFs, with 1.65m in height and weighs in total 50kg. The results can be seen in Figure 4.2. As expected, when trained with a low target velocity ($1m/s$), the character exhibits a walking gait. When trained with a high target velocity ($5m/s$), the character uses a running gait

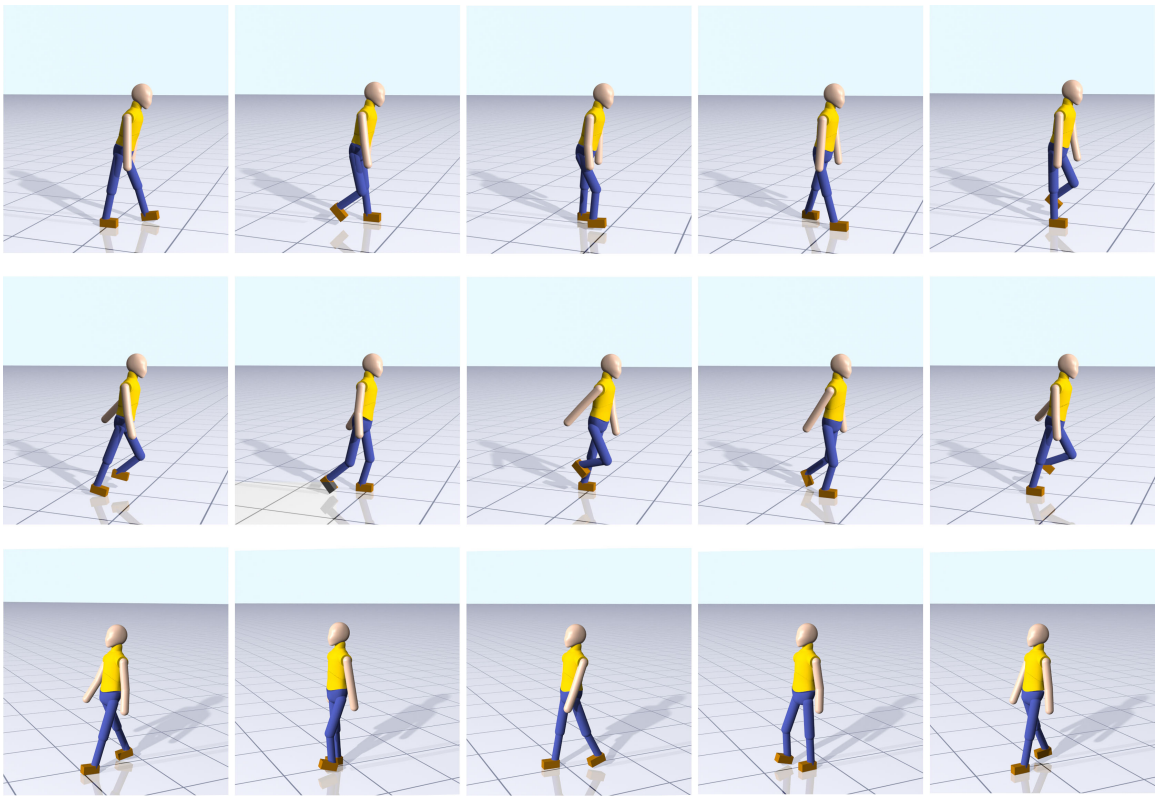


Figure 4.3: Humanoid walking (top), running (middle) and backward walking (bottom). Results are trained using environment-centered curriculum learning and mirror symmetry loss.

indicated by the emergence of a flight phase.

Quadruped Quadrupeds exhibit a large variety of locomotion gaits, such as pacing, trotting, cantering, and galloping. We applied our approach to a quadruped model as shown in Figure4.4(b). The model has 13 links and 22 DOFs, with a height of 1.15m and weight of 88.35kg. As quadrupeds can typically move faster than biped, we trained the quadruped to move at 2m/s and 7m/s. The results are shown in Figure 4.4. The trained policy results in a trotting gait for low target velocity consistently. For high target velocities, the character learns either trotting or galloping, depending on the initial random seed of the policy.

Hexapod We designed a hexapod creature that has 13 links and 24 DOFs, inspired by the body plan of an insect. We trained the hexapod model to move at 2m/s and 4m/s. As shown

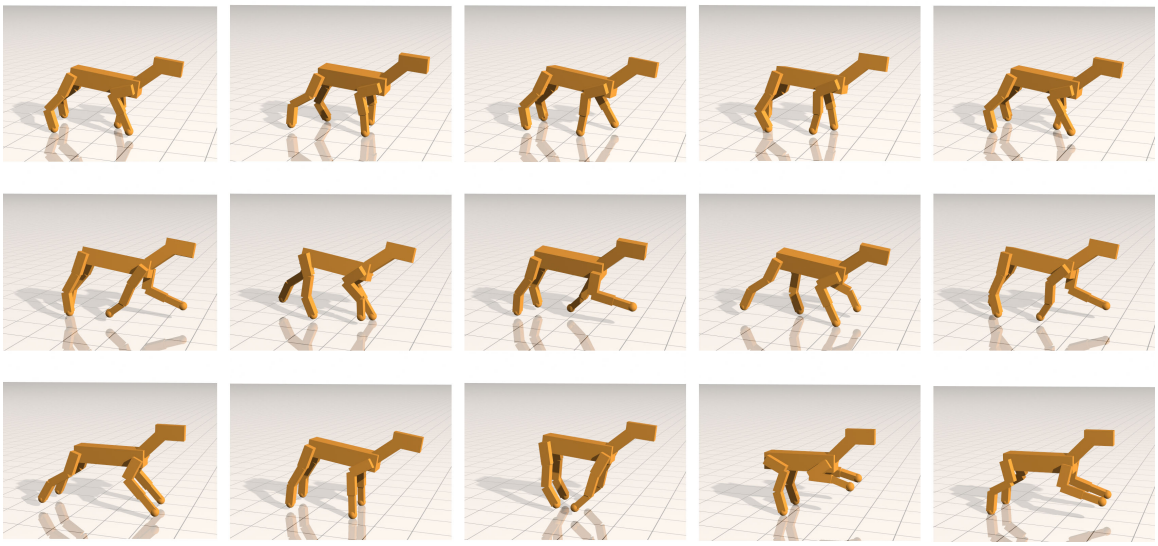


Figure 4.4: Dog trotting (top, middle) and galloping (bottom). Results are trained using environment-centered curriculum learning and mirror symmetry loss.

in Figure 4.5, the hexapod learns to use all six legs to move forward at low velocity, while it lifts the front legs and use the middle and hind legs to 'run' forward at higher velocity.

Humanoid Finally, we trained a locomotion policy for a full humanoid character with a detailed upper body. The character has 13 links and 29 DOFs with 1.75m in height and 76.6kg in weight. We trained the humanoid model to walk at 1.5m/s and run at 5m/s. In addition, we trained the model to walk backward at -1.5m/s. We kept the same reward function parameters between forward and backward walking. Results of the humanoid locomotion can be seen in Figure 4.3. During walking forward and backward, the character learns to mostly relax its arms without much swinging motion. For running, the character learn to actively swing its arms in order to counteract the angular momentum generated by the leg movements, which stabilizes the torso movements during running.

4.5.2 Comparison between Learner-centered and Environment-centered Curriculum Learning

We compare the learner-centered and environment-centered curriculum learning algorithms on the simplified biped model. As demonstrated in the supplementary video, both methods

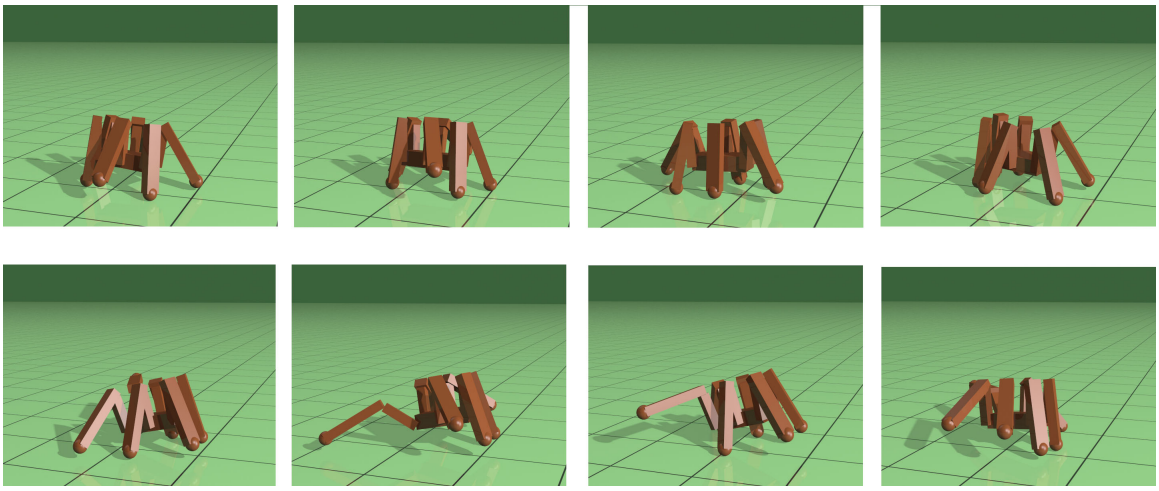
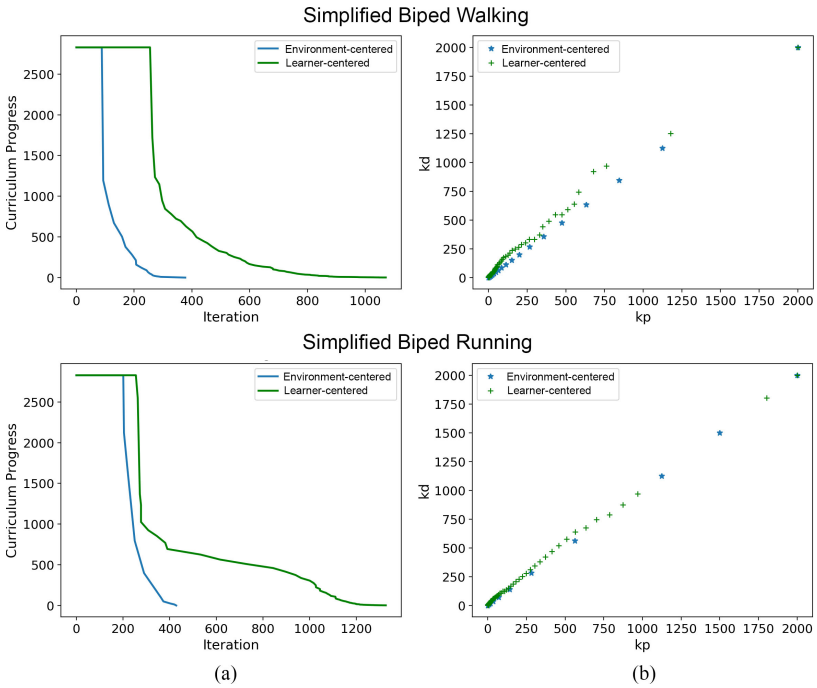


Figure 4.5: Hexapod moving at 2m/s (top) and 4m/s (bottom). Results are trained using environment-centered curriculum learning and mirror symmetry loss.

can successfully train the character to walk and run at target velocities with symmetric gaits. We further analyze the performance of the two algorithms by comparing how they progress in the curriculum space, as shown in Figure 4.6. We measure the progress of the curriculum learning with the $l2$ norm of the curriculum parameter $\mathbf{x}$, since the goal is to reach 0 as fast as possible. We can see that environment-centered curriculum learning shows superior data-efficiency by generating a successful policy with about half the data that is required for the learner-centered curriculum learning.

4.5.3 Comparison with Baseline Methods

To demonstrate the effect of curriculum learning and mirror symmetry loss, we compare our method with environment-centered curriculum learning and mirror symmetry loss (ECL + MSL) to three baseline methods: with environment-based curriculum learning only (ECL), with mirror symmetry loss only (MSL) and using vanilla PPO (PPO) with no mirror symmetry loss nor curriculum learning. The baseline methods are trained on the simplified biped character and the humanoid character for both walking and running. The learning curves for all the tests can be seen in Figure 4.7. In all four of these tasks, our approach learns faster than all of the baseline methods. Without curriculum learning (i.e. blue and



(a) (b)

Figure 4.6: Comparison between environment-centered and learner-centered curriculum learning for simplified biped tasks. (a) Curriculum progress over iteration numbers. 0 in the y axis means no assistance is provided. (b) Points in the curriculum space visited by the two curriculum update schemes.

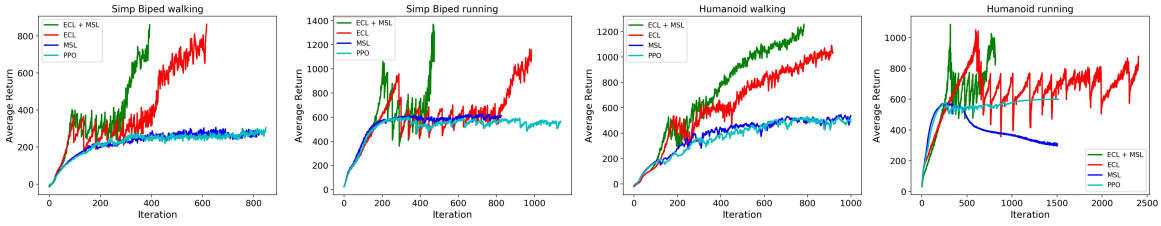


Figure 4.7: Learning curves for the proposed algorithm and the baseline methods.

cyan curves), the algorithm typically learns to either fall slowly or stand still (as is shown in the supplementary video). On the other hand, without mirror symmetry loss, the resulting policy usually exhibits asymmetric gaits and the training process is notably slower, which is mostly evident in the running tasks, as shown in Figure 4.8 and Figure 4.9.

In addition to the three baseline methods described above, we also trained on the simplified biped walking task using vanilla PPO with a modified reward function, where w_e is reduced to 0.1. This allows the character to use higher torques with little penalty ("PPO high torque" in Table 4.2). While the character is able to walk forward without falling, the

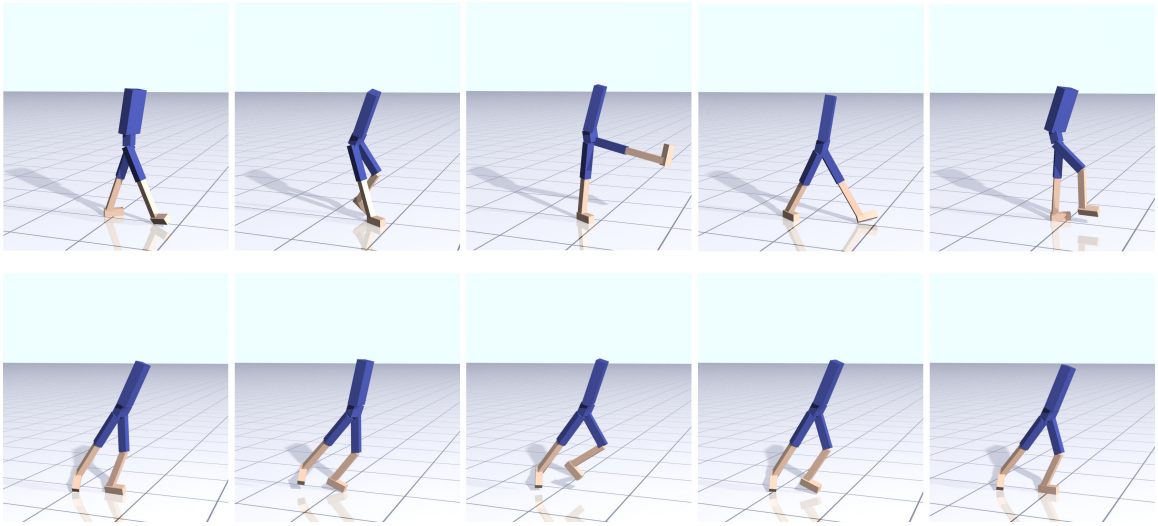


Figure 4.8: Simplified biped walking (top) and running (bottom). Results are trained using environment-centered curriculum learning only (no mirror symmetry loss).

motion appears jerky and uses significantly more torque than our results (see supplementary video).

To compare the policies quantitatively, we report the average actuation magnitude i.e. E_e and use the Symmetry Index metric proposed by Nigg *et al.* to measure the symmetry of the motion [118]:

$$SI(X_L, X_R) = 2\frac{|X_L - X_R|}{X_L + X_R}\%,$$

where X_L and X_R are the average of joint torques produced by the left and right leg respectively. The smaller the value SI is, the more symmetric the gait is. The results can be seen in Table 4.2. As expected, policies trained with our method uses less joint torque and produces more symmetric gaits.

4.5.4 Learning Asymmetric Tasks

One benefit of encouraging symmetric *actions* rather than symmetric *states* is that it allows the motion to appear asymmetric when desired. As shown in Figure 4.10, we trained a humanoid that is walking while holding a heavy object (10kg) in the right hand. The character

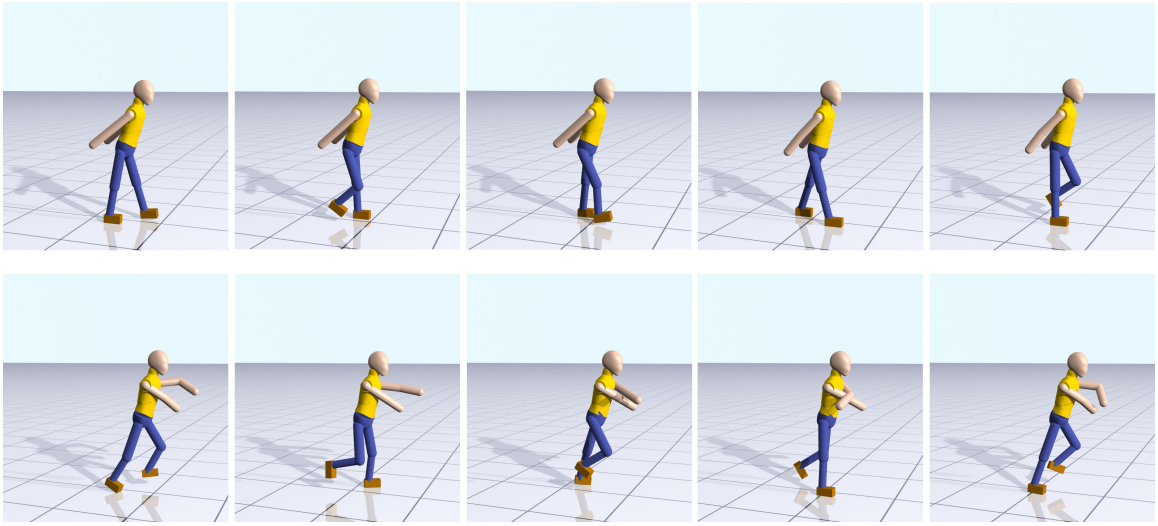


Figure 4.9: Humanoid walking (top) and running (bottom). Results are trained using environment-centered curriculum learning only (no mirror symmetry loss).

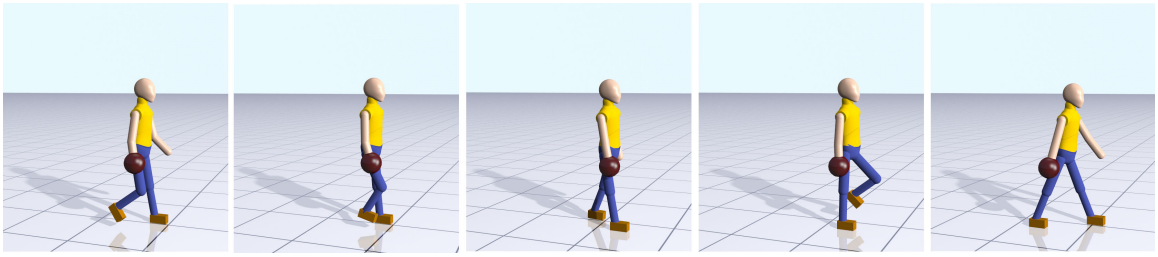


Figure 4.10: Humanoid walking holding heavy object in right hand. Results are trained using environment-centered curriculum learning and mirror symmetry loss.

uses an asymmetric gait that moves more vigorously on the left side to compensate for the heavy object on the right side. If we chose to enforce symmetry on the states directly, the character would likely use a large amount of torque on the right side to make the poses appear symmetric.

4.6 Discussion

In this Chapter, we have demonstrated a reinforcement learning approach for creating low-energy, symmetric, and speed-appropriate locomotion gaits. One element of this approach is to provide virtual assistance to help the character learn to balance and to reach a target speed. The second element is to encourage symmetric behavior through the use of an

Table 4.2: Comparison of trained policies in action magnitude and symmetry. ECL denotes using environment-centered curriculum learning, MSL means mirror symmetry loss and PPO means training with no curriculum learning or mirror symmetry loss. We present results for the successfully trained policies.

Task	Training Setup	E_e	SI
Simplified Biped walk	ECL+MSL	2.01	0.0153
Simplified Biped walk	ECL	2.98	0.1126
Simplified Biped walk	PPO high torque	5.96	0.0416
Simplified Biped run	ECL + MSL	5.57	0.0026
Simplified Biped run	ECL	6.052	0.4982
Humanoid walk	ECL+MSL	6.2	0.0082
Humanoid walk	ECL	7.84	0.0685
Humanoid run	ECL+MSL	17.0976	0.0144
Humanoid run	ECL	18.56	0.0391

additional loss term. When used together, these two techniques provide a method of automatically creating locomotion controllers for arbitrary character body plans. Although our characters demonstrate more natural locomotion gaits comparing to existing work in DRL, the quality of the motion is still not on a par with previous work in computer animation that exploits real-world data. By incorporating biological-based modeling, Jiang *et al.* [119] achieved improved motion quality with reduced requirements for reward engineering. Nevertheless, there is still room for improvement. Understanding and bridging the gap of the perceived naturalness between our approach and real animal and human motions is thus an interesting and important direction to pursue. In addition, our work is only evaluated on terrestrial locomotion with characters represented by articulated rigid bodies. One possible future direction is to apply the curriculum learning to other types of locomotion, such as swimming, flying, or soft-body locomotion.

CHAPTER 5

SIM-TO-REAL TRANSFER OF LOCOMOTION POLICIES

5.1 Motivation

Recent advancements in computer graphics have seen the creation of virtual worlds that are realistic, highly customizable, and highly efficient. However, despite the variety of complex motor skills that computer animation researchers have achieved using modern computer simulation techniques [120, 121, 11, 26] and the notable benefits in autonomy and safety it can potentially deliver, computer simulation has not been embraced in full by robotics researchers. The main reason is that a controller trained in computer simulation usually does not transfer to the real robot due to the discrepancy between the simulation and the real-world, also referred to as the "Reality Gap" in the Evolutionary Robotics community [122, 123]. Researchers have put forth a long list of possible factors that give rise to the Reality Gap, such as simplified dynamic models, inaccurate model parameters, approximated hardware limitations, the absence of uncertainty and latency in sensors and actuators, and other unmodelled factors. Closing the Reality Gap has recently attracted much interest in robotics as the ability to transfer knowledge learned in simulation to the real world can potentially unlock the full capability of deep reinforcement learning for robotic applications.

One of the common techniques to overcome this generalization gap is to train a single policy that can handle a wide range of situations by exposing it to many random scenarios, so-called *domain randomization* (DR) [63]. DR has been successfully demonstrated on sim-to-real transfer problems [58, 61]. However, DR trades optimality for robustness: the policies learned by DR is not optimal under any situation. Furthermore, policies trained with DR is not capable of quickly adapting its own behavior when seeing novel situations,

making it less effective in transferring to environments that are notably different from the training environments. Another popular approach is *meta reinforcement learning* (meta-RL) [77, 71] that aims to solve a new task within a few iterations by training adaptation over a distribution of tasks. However, existing meta-RL methods are mostly effective for adapting to different reward functions, while are in general less effective for adapting to challenging control problems where the dynamics are changed [74].

In this Chapter, we introduce a series of transfer learning algorithms that are geared toward overcoming large discrepancies in dynamics and can achieve near-optimal performance in the target environment. Our core idea is to train a family of policies in randomized simulated environments, each specialized to one or a subset of the training scenarios. We then select the best policy to use when transferring the policies to the target environment. The key to successful sim-to-real transfer lies in two aspects: first, how do we train and represent the family of policies during training? And second, how do we find the best policy to during during transfer? In the following sections, we present our investigations in different possible ways to answer these two questions, and study how the sim-to-real transfer performance is affected by different designs choice within this framework. We will first describe an algorithm that trains a control policy and an online system identification model in order to achieve successful transfer to unknown and changing dynamics (5.2). We then propose the idea of Strategy Optimization (SO) to help the trained policies to overcome larger reality gap by collecting experience in the target environments (5.3). Based on SO, we developed a system for transferring simulation-trained locomotion policies to a real robot, Robotis Darwin OP2 and improved the sample efficiency of the algorithm during transfer (5.4). We further improve the performance of the algorithm by unifying the training and testing process and demonstrated sim-to-real transfer results on a quadruped robot, Ghost Robotics Minitaur (5.5). Figure 5.1 illustrates and contrasts the algorithm schemes of different algorithms described in this Chapter.

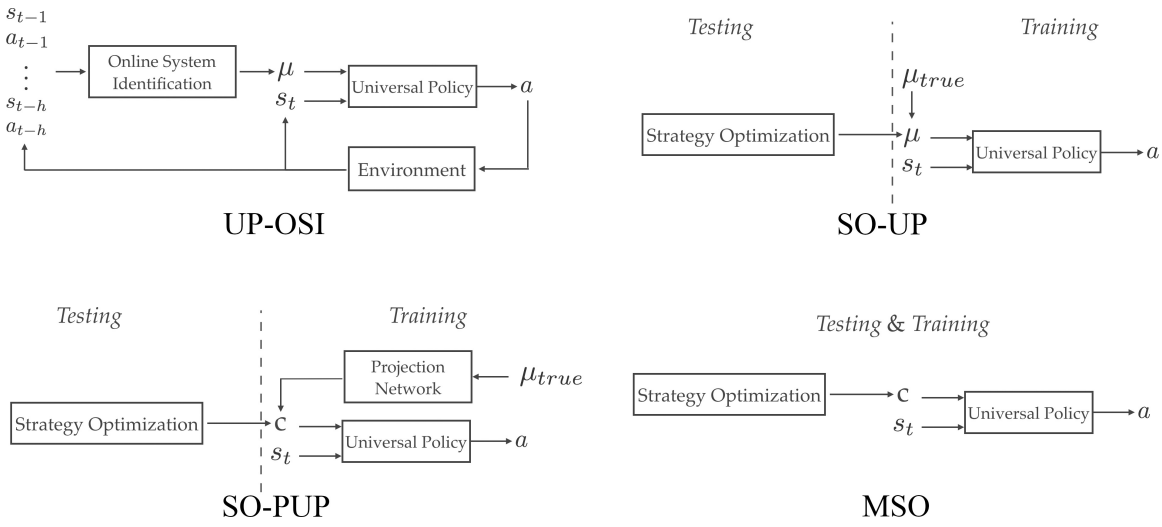


Figure 5.1: Overview of the four algorithms introduced in this Chapter. Top-Left: Universal Policy with Online System Identification (5.2). Top-Right: Strategy Optimization with Universal Policy (5.3). Bottom-Left: Strategy Optimization with Projected Universal Policy (5.4). Bottom-Right: Meta Strategy Optimization (5.5).

5.2 Universal Policy with Online System Identification

5.2.1 Overview

In this section, we introduce our first transfer learning algorithm that learns 1) a universal control policy (UP) that can operate under a space of simulated dynamic models, when provided with the appropriate parameters as input, and 2) an online system identification model (OSI) that predicts the dynamic model parameters given the current state and the recent history of state-action pairs. First, we formulate a reinforcement learning problem to learn a universal policy (UP), $\pi : (\mathbf{o}, \boldsymbol{\mu}) \mapsto \mathbf{a}$, for a space of dynamic models, $\mathbf{s}_{t+1} = \mathcal{P}_{\boldsymbol{\mu}}(\mathbf{s}_t, \mathbf{a}_t)$, parameterized by the dynamic model parameters $\boldsymbol{\mu}$. For partially observable systems, $\mathbf{o}$ contains partial information of $\mathbf{s}$: $\mathbf{o} = g(\mathbf{s})$, while it is identical to $\mathbf{s}$ in fully-observable systems. Second, we formulate a supervised learning problem to train an online system identification model (OSI), $\phi : (\mathbf{o}_{t-h:t}, \mathbf{a}_{t-h:t-1}) \mapsto \boldsymbol{\mu}$, that predicts the dynamic model parameters $\boldsymbol{\mu}$, given the current observation $\mathbf{o}_t$ and the past h time instances of the observation-action pairs. Both components are represented as a standard neural network

and trained offline using simulated data only.

Putting UP and OSI together, at every time instance, we first use OSI (ϕ) to predict the dynamic model parameters $\boldsymbol{\mu}$ based on the current observation of the robot $\mathbf{o}_t$ and the recent history of motion $(\mathbf{o}_{t-1}, \mathbf{a}_{t-1}, \cdots, \mathbf{o}_{t-h}, \mathbf{a}_{t-h})$. Once $\boldsymbol{\mu}$ is identified, we feed both $\boldsymbol{\mu}$ and the current observation $\mathbf{o}_t$ into UP (π) to evaluate the the optimal action $\mathbf{a}_t$ under the predicted dynamic model. We execute $\mathbf{a}_t$ on the robot and push $\mathbf{o}_t$ and $\mathbf{a}_t$ into the history queue. The new state of the system becomes the current observation $\mathbf{o}_t$ and the algorithm advances to the next time step.

UP-OSI is sample-efficient by design because the algorithm does not require real-world samples during offline training. Another important advantage of UP-OSI is that it does not require the model parameters to be identified prior to execution. While some model parameters might not change over time (e.g. mass, length of a body part) and can be identified offline, other parameters related to the unknown environment, such as the friction coefficient of the floor or the mass of objects being manipulated, cannot be easily identified in advance. Part of the power of UP-OSI is that it can dynamically adapt to changing factors in the environment.

We evaluate our method by learning dynamic motor skills and executing them under unknown dynamic models in simulation. In each of the examples, the control policy can successfully execute the task without knowing some crucial parameters of the dynamic model, such as the inertial and geometric parameters of the robot, variable friction coefficients in the environment, and other task-related parameters. Furthermore, we demonstrate that UP-OSI can operate successfully outside the space of dynamic models used for training, as well as under sudden changes in the environment.

5.2.2 Training Universal Policy

Our goal is to learn a control policy that can be generalized to a parameterized space of dynamic models. Many existing methods [124, 125] employ an ensemble approach by learn-

ing a discrete set of control policies and consolidating them into one regression model. Our initial attempt with the ensemble approach showed that, for many dynamic tasks, sometimes a small change in the model parameter requires a drastically different control policy to succeed at the given task. Fitting a regression model to this non-smooth landscape of policies often yields poor generalizability.

In our work, we found that it is possible to directly train a large neural net to represent a universal control policy, $\pi(\mathbf{o}, \boldsymbol{\mu})$, for a space of dynamic models parameterized by $\boldsymbol{\mu}$. With a powerful policy optimization algorithm and sufficient data, the universal policy can achieve high rewards across the space of $\boldsymbol{\mu}$, with comparable performance to policies that have been trained for a specific $\boldsymbol{\mu}$.

For the experiments shown in this section, we use the Trust Region Policy Optimization (TRPO) method [55] to train and show that by simply appending the model parameters $\boldsymbol{\mu}$ to the input state, TRPO can successfully train a universal control policy. However, we need to modify the exploration scheme of TRPO because the part of the state space that represents $\boldsymbol{\mu}$ is not affected by forward simulation when generating rollouts. For each rollout, our algorithm (Algorithm 1) samples $\boldsymbol{\mu}_i$ from a uniform distribution ρ_μ, and generate the motion sequence under the policy $\pi(\mathbf{o}, \boldsymbol{\mu}_i)$ and the dynamic model $\mathcal{P}_{\boldsymbol{\mu}_i}$. Once the state-action pairs are collected in this manner, the update of π follows TRPO exactly.

5.2.3 Learning Online System Identification Model

Even with the ability to perform control under different dynamic models, UP can only succeed at a task when given accurate model parameters, and this information is typically not readily available. We propose to learn an online system identification model (OSI), $\phi : (\mathbf{o}_{t-h:t}, \mathbf{a}_{t-h:t-1}) \mapsto \boldsymbol{\mu}$, that continuously identifies the correct model parameters $\boldsymbol{\mu}$ for UP, when given a short recent history of the states and actions.

The training process can be formulated as a supervised learning problem with the input being a history rollout H and the output being the model parameters $\boldsymbol{\mu}$ under which the

Algorithm 4 Learning UP

Randomly initialize UP network π
for $i = 1 : K$ **do**
    Initialize rollout buffer R
    $\boldsymbol{\mu} \sim \rho_{\mu}$
    $\mathbf{s} \sim p_0$
    while $R.size \leq MaxStep$ **do**
        $\mathbf{o} = g(\mathbf{s})$
        $\mathbf{a} = \pi(\mathbf{o}, \boldsymbol{\mu})$
        $\mathbf{s} = \mathcal{P}_{\boldsymbol{\mu}}(\mathbf{x}, \mathbf{a})$
        $r, terminated = \mathcal{R}(\mathbf{x}, \mathbf{a})$
        Push $(\mathbf{o}, \mathbf{a}, r)$ into R
        if $terminated$ **then**
            $\boldsymbol{\mu} \sim \rho_{\mu}$
            $\mathbf{s} \sim p_0$
    Update π with data in R using DRL algorithms
return π

input rollout is generated:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{(H_i, \boldsymbol{\mu}_i) \subseteq B} \|\phi_\theta(H_i) - \boldsymbol{\mu}_i\|^2 \tag{5.1}$$

where θ are the parameters of the neural net ϕ_θ.

Although the training data can be entirely obtained from simulation, the amount of data can be intractably large to thoroughly cover the input space. Our key observation is that OSI only needs to be accurate for the trajectories that are likely to be observed when performing the tasks of interest. As such, we randomly sample the space of $\boldsymbol{\mu}$ where UP is trained for. For each sampled $\bar{\boldsymbol{\mu}}_i$, we simulate N rollouts using the policy $\pi(\mathbf{o}, \bar{\boldsymbol{\mu}}_i)$ and under the dynamics $\mathcal{P}_{\bar{\boldsymbol{\mu}}_i}$. We then generate short history segments from each rollout and store them in the training buffer B (Line 3-13, Algorithm 2).

After optimizing ϕ using Equation 5.1, we found that the performance of the combined system, UP-OSI, was much worse than simply using UP given true model parameters $\bar{\boldsymbol{\mu}}$. This result is not surprising (retrospectively) because our OSI has only "seen" the motion sequences generated by a control policy $\pi(\mathbf{o}, \bar{\boldsymbol{\mu}})$ under a dynamic model $\mathcal{P}_{\bar{\boldsymbol{\mu}}}$ where their

Algorithm 5 Learning OSI

Randomly initialize OSI network ϕ
Initialize training buffer B
for $i = 1 : K$ **do**
    $\bar{\boldsymbol{\mu}} \sim \rho_\mu$
    for $j = 1 : N$ **do**
        Initialize history queue H
        Fill H by simulating under $\pi(\mathbf{o}, \bar{\boldsymbol{\mu}})$ and $\mathcal{P}_{\bar{\boldsymbol{\mu}}}$
        for $t = 0 : T - 1$ **do**
            Pop H
            $\mathbf{a}_t = \pi(\mathbf{o}_t, \bar{\boldsymbol{\mu}})$
            $\mathbf{s}_{t+1} = \mathcal{P}_{\bar{\boldsymbol{\mu}}}(\mathbf{o}_t, \mathbf{a}_t)$
            Push $(\mathbf{o}_{t+1}, \mathbf{a}_t)$ in H
            Store $(H, \bar{\boldsymbol{\mu}})$ in B
Optimize ϕ using data in B
while not converge **do**
    for $i = 1 : K$ **do**
        $\bar{\boldsymbol{\mu}} \sim \rho_\mu$
        for $j = 1 : N$ **do**
            Initialize history queue H
            Fill H by simulating under $\pi(\mathbf{o}, \bar{\boldsymbol{\mu}})$ and $\mathcal{P}_{\bar{\boldsymbol{\mu}}}$
            for $t = 0 : T - 1$ **do**
                $\hat{\boldsymbol{\mu}} = \phi(H)$
                Pop H
                $\mathbf{a}_t = \pi(\mathbf{o}_t, \hat{\boldsymbol{\mu}})$
                $\mathbf{s}_{t+1} = \mathcal{P}_{\bar{\boldsymbol{\mu}}}(\mathbf{s}_t, \mathbf{a}_t)$
                Push $(\mathbf{o}_{t+1}, \mathbf{a}_t)$ in H
                Store $(H, \bar{\boldsymbol{\mu}})$ in B
    Optimize ϕ using data in B
return ϕ

model parameters are consistent. In other words, all the training examples so far only cover the "good cases" where the control policy is operating optimally under a given dynamic model. When we tested OSI with an unseen initial sequence, OSI was likely to make some error in the prediction. This error is exacerbated because the next sequence that OSI will see is generated by a control using an erroneously predicted $\hat{\boldsymbol{\mu}}$ under the true model parameters $\mathcal{P}_{\bar{\boldsymbol{\mu}}}$, where $\hat{\boldsymbol{\mu}} \neq \bar{\boldsymbol{\mu}}$.

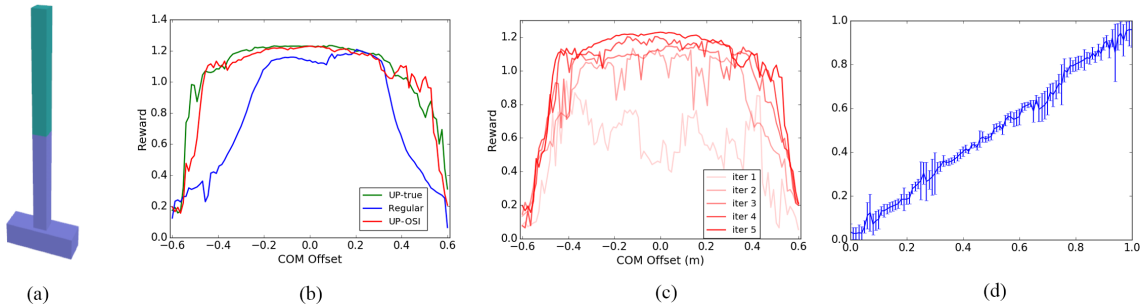
Our solution is to iteratively improve OSI by introducing "bad cases" with mismatched model parameters used for control ($\pi(\mathbf{o}, \hat{\boldsymbol{\mu}})$) and for forward simulation ($\mathcal{P}_{\bar{\boldsymbol{\mu}}}$). For each

iteration, we generate more training examples using the current OSI and UP. We randomly sample in the space of $\boldsymbol{\mu}$ and generate rollouts like before. However, we feed the $\boldsymbol{\mu}$ predicted by the current OSI into UP, instead of the true model parameters $\bar{\boldsymbol{\mu}}$ used for forward simulation. Note that in Line 25 of Algorithm 2, the dynamic model has the parameter $\bar{\boldsymbol{\mu}}$ which is different from the one used for the control (Line 22, 24). Mixing the mismatched training examples with previously generated ones, we train OSI again using Equation 5.1. After a small number of iterations (3-5, see Section IV), the performance of UP-OSI becomes close to the performance of UP that is provided with the true model parameters.

5.2.4 Results

We evaluate UP-OSI on four motor control problems. In each example, the control policy does not know the true model parameters in advance and relies on OSI to identify the parameters during execution. We vary different model parameters, such as mass, friction coefficient, or task-related parameters to demonstrate that UP-OSI can successfully perform all the motor skills under unknown dynamic models. We compare the performance of UP-OSI against the performance of the condition, *UP-true*, which uses UP given the true model parameters. The performance of UP-true can be regarded as an informal upper bound for UP-OSI. We also evaluate the performance of using different motion history sizes for one of the examples.

All results presented in this work are simulated in PyDart2 [116], a python wrapper for DART [5], which is a multibody physics simulator supported by Gazebo. The simulation timestep is set to $0.002s$. For UP, we use a feedforward neural network with two hidden layers, comprised of 64 units in both hidden layers with tanh activation function, followed by a linear fully connected final layer. For OSI with motion history shorter than 5 steps, we use three hidden layers, with 256, 128, and 64 hidden units. For longer motion history, we use a three layer LSTM network, with 64, 64, and 32 hidden units. Both architectures uses tanh activation function and we add a dropout layer for OSI after each hidden layer with a



(a) (b) (c) (d)

Figure 5.2: Results for the Double Inverted Pendulum Task. (a) Illustration of the task. (b) Performance of UP-true, "Regular" Controller and UP-OSI. The horizontal axis is the model parameter (center of mass), and the vertical axis is the performance. (c) The evolution of optimizing UP-OSI. The reward across the training range of μ increases iteratively. (d) Mean and standard deviation of the predicted model parameter. The x-axis indicates the true model parameters while the y-axis indicates the predicted ones by OSI. The model parameters have been normalized to be in $[-1, 1]$.

dropout rate of 0.1.

The learning process for UP takes 500 iterations of TRPO updates. The amount of data collected during each iteration varies by the difficulty of the tasks. We run five iterations for training OSI. At each iteration we sample 30 different $\boldsymbol{\mu}$ values and collect 5 seconds for each $\boldsymbol{\mu}$. We use a motion history of 3 for our examples except for the single-legged robot locomotion, for which we used a longer motion history.

Double inverted pendulum with unknown center of mass

We begin with a classic motor control problem: balancing a double inverted pendulum that is mounted on a movable cart. We define the reward function as,

$$\mathcal{R}(\mathbf{s}) = -k_1(\sigma_1 + \sigma_2)^2 - k_2|p_{cart}| + 10,$$

where σ_1 and σ_2 are angles of the two poles from the upright configuration, p_{cart} is the position of the cart and k_1 and k_2 are the corresponding weights of the two terms. We normalize the angles to be in $[0, \pi]$ and use $k_1 = 10.0, k_2 = 1.0$ in our experiment. The length of the two poles are both $0.5m$. We terminate the simulation when $|p_{cart} \geq 5|$ or $(\sigma_1 + \sigma_2) \geq 0.5\pi$.

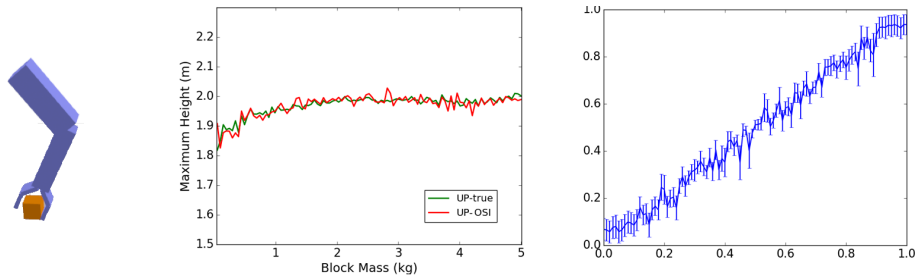
The unknown model parameter for this problem is the center of mass of the lower pole, which has an unknown offset, $(\mu, 0.2\mu)$, from the geometric center. To ensure that the control policy would need to apply different strategies to balance the pendulum when different model parameters are given, we allow the offset to vary across a wide range: $\mu \in [-0.6m, 0.6m]$. Note that the purpose of the vertical offset (0.2μ) is to break the symmetry of the problem to further increase the difficulty of control.

At each training iteration of UP, we collect $150,000$ samples using the physics simulator. Figure 5.2(b) shows the normalized performance of the trained UP-OSI across different $\boldsymbol{\mu}$ values, comparing against UP-true (the informal upper bound). The performance of each $\boldsymbol{\mu}$ in Figure 5.2(b) is the normalized average accumulated reward of 20 rollouts starting from a randomly perturbed initial state. If the performance value is above 1.0, the double inverted pendulum is able to balance. We also compare UP-OSI to a policy with conventional state input and control output but trained by data simulated from a range of model parameters (denoted as "regular" in Figure 5.2(b)). The purpose of this comparison is to show that providing the model parameters as input to UP results in a more powerful control policy under a range of dynamic models,

To demonstrate the learning process of OSI network, we plot the same reward-model parameter graph at each iteration of training. As shown in Figure 5.2(c), the performance of UP-OSI improves over time and approaches the performance of UP-true. In Figure 5.2(d), we plot the mean and the standard deviation for the model parameter identified by the trained OSI for each ground truth μ on x-axis. This shows that OSI is indeed able to identify the model parameter in this task.

Manipulator with unknown object mass

In this example, we train a robot arm to grab a block and throw it up to a certain height but not beyond. The arm is initially pointing down and the block is in the air near the gripper of the arm. Similar motor skill can be observed in the serving of a tennis ball.



(a) (b) (c)

Figure 5.3: Results for the Robot Arm object throwing task. (a) Illustration of the Task. (b) Performance of UP-true and UP-OSI. The horizontal axis is the model parameter (mass of the object), and the vertical axis is the performance (maximum height of the object). (c) Mean and standard deviation of the predicted model parameter. The x-axis indicates the true model parameters while the y-axis indicates the predicted ones by OSI. The model parameters have been normalized to be in $[-1, 1]$.

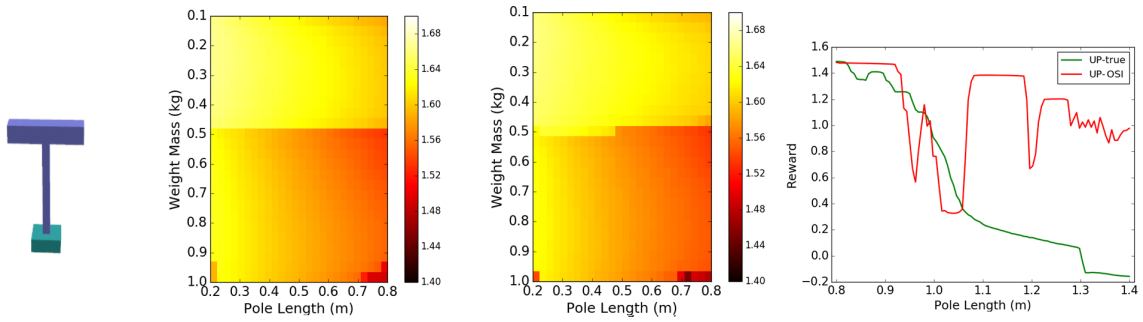
The observation $\mathbf{o}$ includes the joint position $\mathbf{q}$, joint velocity $\dot{\mathbf{q}}$ of the robot arm, and the position of the block $\mathbf{p}_{block}$. The reward function is defined as:

$$\mathcal{R}(\mathbf{s}, \mathbf{a}) = -k_1 r_h - k_2 ||\mathbf{a}||^2 - k_3 ||\dot{q}||^2 + 35$$

$$r_h = \begin{cases} h_{target} - h_{block}, & \text{if } h_{block} \leq h_{target} \\ 0, & \text{otherwise} \end{cases},$$

where $k_1 = 10, k_2 = 1e-5, k_3 = 1e-3, h_{target} = 2m$ and h_{block} is the height of the block. We terminate the rollout when the box falls below $-0.2m$ or when the block is more than $0.8m$ away horizontally. By giving zero reward beyond h_{target}, we encourage the robot arm to throw the block in a way that it has low velocity when it reaches h_{target}, such that it can stay in the high reward region as long as possible. The unknown model parameters is the mass of the block. The robot needs to infer the weight of the block and use the proper amount of effort to throw it up to the correct height.

During the training of UP, we collect $50,000$ samples for each iteration. The performance of UP-OSI and UP-true is plotted in Figure5.3(b). We measure the performance by the highest point reached by the block. The closer to $h_{target} = 2m$, the better the performance. We also plot the mean and standard deviation of the predicted block mass



(a) (b) (c) (d)

Figure 5.4: Results for the Cart-Pole Swing-Up task. (a) Illustration of the task. (b) Performance of UP-true visualized in the domain of pole length and weight mass. (c) Performance of UP-OSI. (d) Performance with model parameters that exceed the training range by 100%.

throughout the test, as shown in Figure5.3(c).

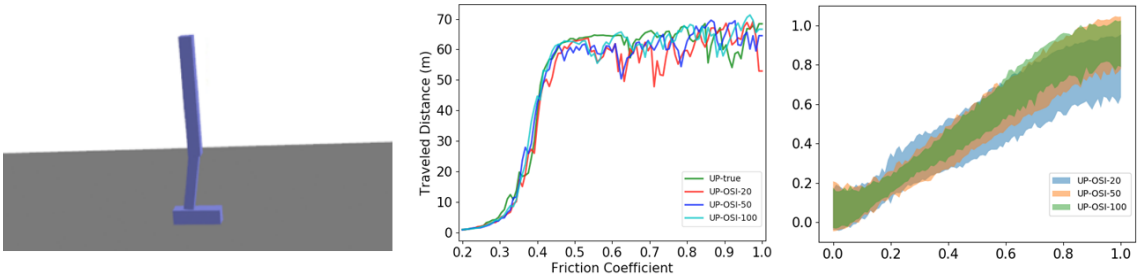
Cart-pole swing-up with unknown pole length and unknown attached mass

To solve the classic cart-pole swing-up problem, the control policy needs to learn not only how to balance the pole, but also how to swing it up from a straight down position. Our experiment makes two modifications to increase the difficulty of the problem. First, we limit the force used by the cart to be within $[-40N, 40N]$. As such, the controller must swing the pole back and forth before it rises up. We also attach an additional mass to the tip of the pole to mimic the weight lifting task (Figure 5.4(a)).

We use a variant of the reward function suggested by [126]: $r_\sigma = w\sigma^2 + v\log(\sigma^2 + a)$, where σ is the angle of the pole. The first term encourages fast learning of swing-up motion and the second term encourages fast learning of balance. In our experiment, we set $w = 1, v = 1, a = 0.1$. Similar to the double inverted pendulum task, we also add a term to encourage the cart to stay at the center of the track, $r_{cart} = |p_{cart}|$. Together, our reward function is defined as:

$$\mathcal{R}(\mathbf{s}) = -k_1 r_\sigma - k_2 r_{cart} + 10.0,$$

where $k_1 = 1.0, k_2 = 0.2$. In our setup, the pole has 0 position when it is upright. During

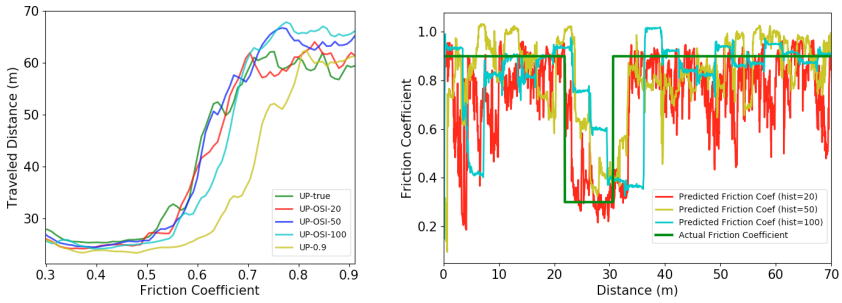


(a) (b) (c)

Figure 5.5: Results for the Hopper task. (a) Illustration of the task. (b) Performance of UP-true and UP-OSI. The horizontal axis is the model parameter (friction coefficient), and the vertical axis is the performance (maximum distance traveled in $1,000$ simulation steps). (c) Mean and standard deviation of the predicted model parameter with different motion history sizes. The x-axis indicates the true model parameters while the y-axis indicates the predicted ones by OSI. The model parameters have been normalized to be in $[-1, 1]$.

the simulation, we randomly initialize the position of the pole to be either π or $-\pi$ with a small noise drawn from $\mathcal{N}(0, 0.005)$. We terminate the rollout when the pole rotates more than 4π from the initial position, or when the cart is more than $2m$ from the center.

The unknown model parameters in this example includes the additional mass attached to the top of the pole ($\boldsymbol{\mu}_{mass} \in [0.1kg, 1.0kg]$), and the length of the pole ($\boldsymbol{\mu}_{length} \in [0.2m, 0.8m]$). To closely compare with the cart-pole examples in [127], where they control an inverted pendulum with varying pole length using a RNN with the whole history trajectory as input, we train OSI to estimate the velocity of the system, instead of directly giving the true velocity to the policy as part of the state. As such, the space of model parameters for this example is $\mathbb{R}^4$. At each iteration of UP training, we run $70,000$ samples. We normalize the resulting reward such that if a policy achieves averaged accumulated reward of more than one, then it usually can swing up and balance the cart-pole system. Figure 5.4(b) and (c) show that UP-OSI can achieve high reward for a range of unknown pole lengths, similar to the inverted pendulum result shown in [127], but UP-OSI only requires three time steps of history as input. In addition, the mass attached to the tip is also an unknown that needs to be simultaneously identified with the pole length.



(a) (b)

Figure 5.6: Results for testing the adaptability of UP-OSI trained for the Hopper task. (a) Performance of varying contact friction test. We tested UP-true, UP-OSIs and UP with input friction coefficient fixed at 0.9. A low-pass filter has been applied for better visualization. (b) OSI-predicted and actual friction coefficient in varying contact friction test.

Generalization beyond training range

One important aspect of the policy generalizability is whether it works with model parameters that were not seen during the training phase. We perform such test on the cart-pole swing up problem with a pole length range of $[0.8m, 1.4m]$, which is 100% beyond the original training range. We also linearly increase the the attached mass with the pole length. We test both UP-true and UP-OSI with this extended range, which is shown in Figure 5.4(d). The result shows that UP-OSI can work for a large range of unknown pole length and attached mass with only position information as input. More interestingly, UP-OSI significantly outperforms UP-true in this range of $\boldsymbol{\mu}$ unseen during training.

Hopper with unknown friction coefficient

Correctly identifying contact information is crucial to many locomotion tasks. In this example, we demonstrate that our method can be applied to identify the friction coefficient at the contact point in an online fashion. The task is to control a single leg robot in 2D, the Hopper, to hop forward as fast as possible without falling. The reward is defined as

$$r(\mathbf{s}, \mathbf{a}) = k_1 \dot{x} - k_2 ||\mathbf{a}||^2 + 2.0,$$

where $\dot{x}$ is the forward velocity of the torso and $k_1 = 1, k_2 = 0.003$ for our experiments. The unknown model parameter is the friction coefficient with the range $\boldsymbol{\mu} \in [0.2, 1.0]$. We plot the distance traveled by the hopper before the termination criteria is satisfied (the hopper falls or the maximum length of the rollout is reached) instead of the reward value as our focus is on robot locomotion. The input to UP include the joint position of the hopper $\mathbf{q}$, the joint velocity $\dot{\mathbf{q}}$ and the friction coefficient μ between the foot and the ground. Note that we don't use position in the forward direction in the input state, because it is not directly related to the task.

We use $75,000$ samples each iteration during the training of UP. As the agent can only infer the friction coefficient when the foot is in contact, we use a longer motion history in OSI. We test a motion history of 20, 50 and 100 steps. Figure 5.5(b) shows the performance of all three UP-OSIs compared to UP-true. Due to the difficulty of the task, UP can only perform well around $\mu = [0.5, 1.0]$. However, being able to identify friction coefficients in this range, i.e. between the coefficient for wood-concrete contact and that for rubber-concrete contact, is sufficient for most practical applications. Figure 5.5(c) shows the mean and standard deviation of the predicted model parameter during the test at each $\boldsymbol{\mu}$. We can observe that a longer motion history leads to a more precise estimation of the model parameter.

Generalization to a varying model parameter

We run the trained UP-OSI for Hopper on a track with varying friction coefficients to test its generalizability. We create a track with friction coefficient $\boldsymbol{\mu}_{const} = 0.9$ everywhere except for the region between $20m$ to $30m$. We then vary the friction coefficient $\boldsymbol{\mu}_{vary}$ in this region and plot the performance of the controller with regard to $\boldsymbol{\mu}_{vary}$. Figure 5.6(a) shows the performance of UP-OSI and the UP-true. Note that UP-true was given the ground truth friction coefficient at each time instance as if it has a perfect contact-friction sensor on the foot, while UP-OSI needs to identify this information based on the recent history of the

motion. We test three motion history sizes (20, 50 and 100) and the results show that UP-OSI can achieve comparable and sometimes better performance than UP-true. We also test the performance of UP with fixed input $\boldsymbol{\mu} = 0.9$, i.e. the hopper is unaware of the change in friction coefficient, shown as the yellow curve in Figure 5.6(b). The poorer performance shows that it is crucial to detect the varying friction coefficient in order to succeed in this task.

In Figure 5.6(b), we plot the friction coefficient predicted by OSI over time for a track with $\boldsymbol{\mu}_{vary} = 0.3$. We show that OSI with all three motion history sizes can identify the changes in model parameter during the task. We also observe that the predicted parameter is more noisy with a shorter motion history size, while the adaptation to the change in the model parameter is slower with a longer motion history size. Note that we did not provide any training examples with temporally-varying μ when training either UP-true or UP-OSI networks.

Comparison to End-to-End Training

One of the core ideas in our approach is to decompose the controller into UP and OSI such that OSI can be trained efficiently using supervised learning. To show the advantage in such decomposition, we compare our method to an End-to-End training approach for the hopper example.

We use the TRPO algorithm implemented in RLLAB [110] to directly train a control policy represented by an LSTM network with one hidden layer of 64 units. We train the LSTM policy with the same batch size as in training universal policy and run it for 500 iterations. The result can be found in Figure 5.7. With comparable amount of training data, our approach can achieve a notably better performance than an End-to-End approach.

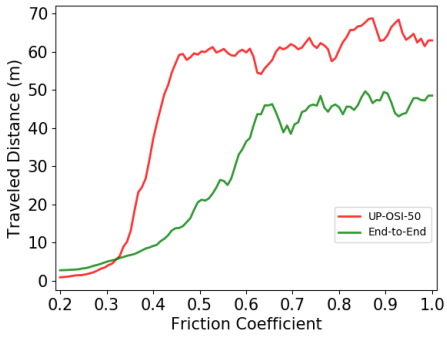


Figure 5.7: Comparison between our approach and End-to-End trained policy on the single-legged robot task.

5.2.5 Discussions

We have demonstrated that UP-OSI can learn successful policies that operate in unknown and changing dynamics. Nevertheless, we recognize a few limitations of this algorithm. First, identifying high-dimensional model parameters remains an unsolved challenge. Although we demonstrated that OSI can perform well for model parameters in $\mathbb{R}^4$ (the cart-pole example), the sample-efficiency and the performance will likely drop with the increase of observation space and the dimensions of the dynamics parameters $\boldsymbol{\mu}$. Second, during training, OSI has only been exposed to transition data generated by the training dynamics. As a result, the estimation accuracy will drop as the testing dynamics gets farther away from the training ones. More importantly, when the testing dynamics cannot be represented by any of the training variations (e.g. due to unmodelled effects), it is not clear what should OSI output. In the next section, we introduce a new algorithm that allows us to overcome these challenges and achieve better transfer performance in a sim-to-sim transfer setting.

5.3 Universal Policy with Strategy Optimization

5.3.1 Overview

As discussed in the previous section (5.2.5), a core assumption that UP-OSI makes is that OSI model can produce reasonable estimation of the dynamics parameters from the history

data. When the testing dynamics cannot be well approximated by the training dynamics (e.g. due to unmodelled effects), it becomes unclear what is the best estimation OSI should produce. Such situation is inevitable when dealing with the sim-to-real transfer problem as modeling all aspects of the real-world dynamics faithfully is infeasible. In this section, we introduce the idea of strategy optimization (SO) that enables us to overcome larger reality gaps, even for testing environments that are notably different from the training variations. The core idea of SO is that, instead of relying on an identification model trained in simulation to select the policy to use during testing, we directly optimize for the best policy in the testing environment based on the task performance.

Our algorithm can be divided to two stages. The first stage trains a universal policy (UP), $\pi : (\mathbf{o}, \boldsymbol{\mu}) \mapsto \mathbf{a}$, as described in Section 5.2.2. In the second stage we perform a search over the space of $\boldsymbol{\mu}$ in the target environment to find the one that achieves the highest task performance.

We evaluate our method on three examples that demonstrate transfer of a policy learned in one simulator DART, to another simulator MuJoCo. Due to the differences in the constraint solvers, these simulators can produce notably different simulation results. A more detailed description of the differences between DART and MuJoCo is provided in Appendix A. We also add latency to the MuJoCo environment to mimic a real world scenario, which further increases the difficulty of the transfer. In addition, we use a quadruped robot simulated in Bullet to demonstrate that our method can overcome actuator modeling errors. Latency and actuator modeling have been found to be important for Sim-to-Real transfer of locomotion policies [58, 10]. Finally, we transfer a policy learned for a robot composed of rigid bodies to a robot whose end-effector is deformable, demonstrating the possiblity of using our method to transfer to problems that are challenging to model faithfully.

5.3.2 Strategy Optimization

During strategy optimization (SO), we search for the optimal strategy in the space of $\boldsymbol{\mu}$ for the target environment. In other word, we solve the following optimization problem:

$$\boldsymbol{\mu}^* = \underset{\boldsymbol{\mu}}{\operatorname{argmax}}\, J_{\mathcal{M}^r}(\pi_{\boldsymbol{\mu}}). \tag{5.2}$$

Solving Equation 5.2 can be done efficiently because the search space in Equation 5.2 is the space of dynamic parameters $\boldsymbol{\mu}$, rather than the space of policies, which are represented as neural networks in our implementation. There are different possible way to to solve Equation 5.2, such as Bayesian optimization, model-based methods or evolutionary algorithms. In this section, we use an evolutionary algorithm, Covariance Matrix Adaptation (CMA) [80], to perform the optimization. We found that for the search space that are relatively higher (¿ 5D), CMA works more reliably than alternative methods, while for lower dimensional search space, Bayesian Optimization (BO) achieves better sample efficiency. We will discuss in the next section how to leverage this observation to further improve our algorithm. For model-based methods, we find that they achieve comparable transfer performance to CMA-ES, while require knowing full state of the robot. At each iteration of CMA, a set of samples are drawn from a Gaussian distribution over the space of $\boldsymbol{\mu}$. For each sample, we instantiate a strategy $\pi_{\boldsymbol{\mu}}$ and use it to generate rollouts in the target environment. The fitness of the sample is determined by evaluating the rollouts using $J_{\mathcal{M}^r}$. Based on the fitness values of the samples in the current iteration, the mean and the covariance matrix of the Gaussian distribution are updated for the next iteration.

5.3.3 Results

To evaluate the ability of our method to overcome the reality gap, we train policies for four locomotion control tasks (hopper, walker2d, half cheetah, quadruped robot) and transfer each policy to environments with different dynamics. To mimic the reality gap seen

in the real-world, we use target environments that are different from the source environments in their contact modeling, latency or actuator modeling. In addition, we also test the ability of our method to generalize to discrepancies in body mass, terrain slope and end-effector materials. Figure 5.8 shows the source and target environments for all the tasks and summarizes the modeled reality gap in each task. During training, we choose different combinations of dynamic parameters to randomize and make sure they do not overlap with the variations in the testing environments. For clarity of exposition, we denote the dimension of the dynamic parameters that are randomized during training as $dim(\mu)$. For all examples, we use the Proximal Policy Optimization (PPO) [56] to optimize the control policy. A more detailed description of the experiment setup as well as the simulated reality gaps are provided in Appendix B. For each example presented, we run three trials with different random seeds and report the mean and one standard deviation for the total reward.

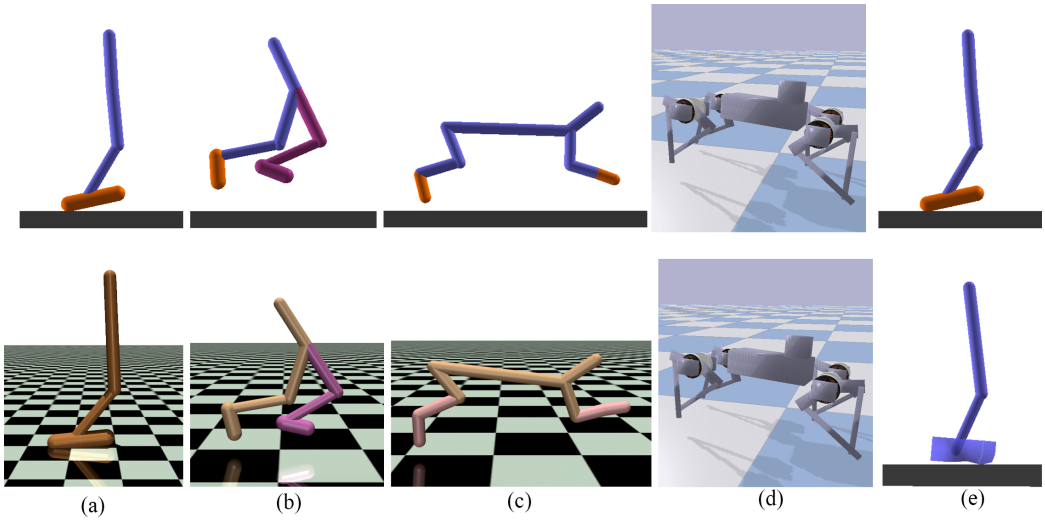


Figure 5.8: The environments used in our experiments. Environments in the top row are source environments and environments in the bottom row are the target environments we want to transfer the policy to. (a) Hopper from DART to MuJoCo. (b) Walker2d from DART to MuJoCo with latency. (c) HalfCheetah from DART to MuJoCo with latency. (d) Minitaur robot from inaccurate motor modeling to accurate motor modeling. (e) Hopper from rigid to soft foot.

5.3.4 Baseline Methods

We compare our method, Strategy Optimization with CMA-ES (SO-CMA) to three baseline methods: training a robust policy (Robust), training an adaptive policy (Hist) and training a Universal Policy with Online System Identification (UP-OSI) as in the previous section. The robust policy is represented as a feed forward neural network, which takes as input the most recent observation from the robot, i.e. $\pi_{robust} : \mathbf{o} \mapsto \mathbf{a}$. The policy needs to learn actions that work for all the training environments, but the dynamic parameters cannot be identified from its input. In contrast, an adaptive policy is given a history of observations as input, i.e. $\pi_{adapt} : (\mathbf{o}_{t-h}, \ldots, \mathbf{o}_t) \mapsto \mathbf{a}_t$. This allows the policy to potentially identify the environment being tested and choose the actions based on the identified environment. There are many possible ways to train an adaptive policy, for example, one can use an LSTM network to represent the policy or use a history of observations as input to a feed-forward network. We find that for the tasks we demonstrate, directly training an LSTM policy using PPO is much less efficient and reaches lower end performance than training a feed-forward network with history input. Therefore, in our experiments we use a feed-forward network with a history of 10 observations to represent the adaptive policy π_{adapt}. We also compare our method to UP-OSI. For fair comparison, we continue to train the baseline methods after transferring to the target environment, using the same amount of samples SO-CMA consumes in the target environment. We refer this additional training step as 'fine-tuning'. In addition to the baseline methods, we also compare our method to the performance of policies trained directly in the target environments, which serves as an 'Oracle' benchmark. The Oracle policies for Hopper, Walke2d, HalfCheetah and Hopper Soft was trained for $1,000,000$ samples in the target environment as in Schulman et al. [56]. For the quadruped example, we run PPO for $5,000,000$ samples, similar to Tan et al. [58]. For fine-tuning of the Robust and Adaptive policy in the target environment, we sample $2,000$ steps from the target environment at each iteration of PPO, which is the default value used in OpenAI Baselines. In the case where we use a maximum of $50,000$ samples

for fine-tuning, this amounts to 50 iterations of PPO updates.

5.3.5 Hopper DART to MuJoCo

In the first example, we build a single-legged robot in DART similar to the Hopper environment simulated by MuJoCo in OpenAI Gym [128]. We investigate two questions in this example: 1) does SO-CMA work better than alternative methods in transferring to unknown environments? and 2) how does the choice of $dim(\mu)$ affect the performance of policy transfer? To this end, we perform experiments with $dim(\mu) = 2$, 5 and 10. For the experiment with $dim(\mu) = 2$, we randomize the mass of the robot's foot and the restitution coefficient between the foot and the ground. For $dim(\mu) = 5$, we in addition randomize the friction coefficient, the mass of the robot's torso and the joint strength of the robot. We further include the mass of the rest two body parts and the joint damping to construct the randomized dynamic parameters for $dim(\mu) = 10$. The specific ranges of randomization are described in Appendix B.

We first evaluate how the performance of different methods varies with the number of samples in the target environment. As shown in Figure 5.9, when $dim(\mu)$ is low, none of the four methods were able to transfer to the MuJoCo Hopper successfully. This is possibly due to there not being enough variation in the dynamics to learn diverse strategies. When $dim(\mu) = 5$, SO-CMA can successfully transfer the policy to MuJoCo Hopper with good performance, while the baseline methods were not able to adapt to the new environment using the same sample budget. We further increase $dim(\mu)$ to 10 as shown in Figure 5.9 (c) and find that SO-CMA achieved similar end performance to $dim(\mu) = 5$, while the baselines do not transfer well to the target environment.

We further investigate whether SO-CMA can generalize to differences in joint limits in addition to the discrepancies between DART and MuJoCo. Specifically, we vary the magnitude of the ankle joint limit in $[0.5, 1.0]$ radians (default is 0.785) for the MuJoCo Hopper, and run all the methods with $30,000$ samples. The result can be found in Figure

5.10. We can see a similar trend that with low $dim(\mu)$ the transfer is challenging, and with higher value of $dim(\mu)$ SO-CMA is able to achieve notably better transfer performance than the baseline methods.

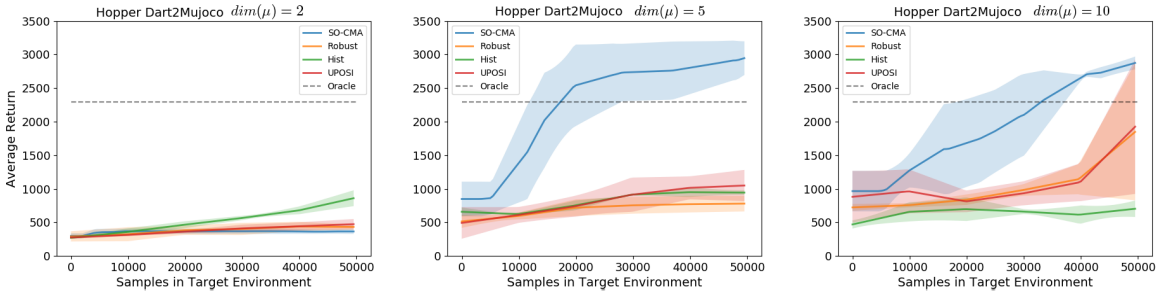


Figure 5.9: Transfer performance vs Sample number in target environment for the Hopper example. Policies are trained to transfer from DART to MuJoCo.

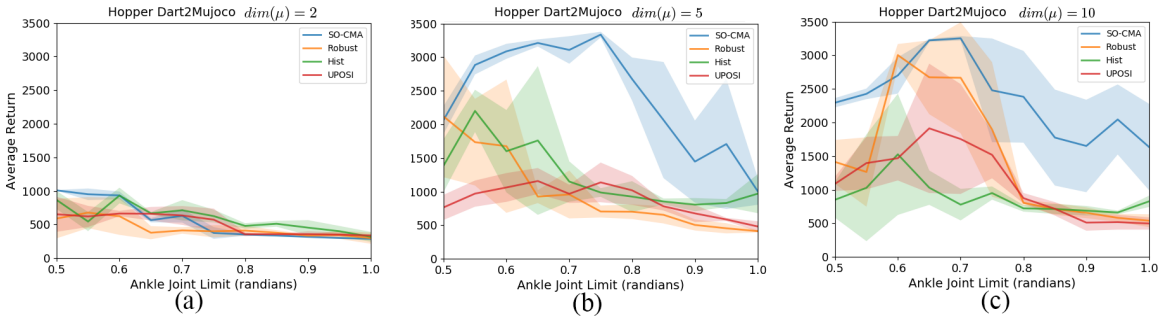


Figure 5.10: Transfer performance for the Hopper example. Policies are traiend to transfer from DART to MuJoCo with different ankle joint limits (horizontal axis). All trials run with total sample number of $30,000$ in the target environment.

5.3.6 Walker2d DART to MuJoCo with latency

In this example, we use the lower body of a biped robot constrained to a $2D$ plane, according to the Walker2d environment in OpenAI Gym. We find that with different initializations of the policy network, training could lead to drastically different gaits, e.g. hopping with both legs, running with one legs dragging the other, normal running, etc. Some of these gaits are more robust to environment changes than others, which makes analyzing the performance of transfer learning algorithms challenging. To make sure the policies are more

comparable, we use the symmetry loss from Yu et al. [11], which leads to all policies learning a symmetric running gait. To mimic modeling error seen on real robots, we add a latency of 8ms to the MuJoCo simulator. We train policies with $dim(\mu) = 8$, for which we randomize the friction coefficient, restitution coefficient and the joint damping of the six joints during training. Figure 5.11 (a) shows the transfer performance of different method with respect to the sample numbers in the target environment.

We further vary the mass of the robot's right foot in $[2, 9]$kg in the MuJoCo Walker2d environment and compare the transfer performance of SO-CMA to the baselines. The default foot mass is 2.9 kg. We use in total $30,000$ samples in the target environment for all methods being compared and the results can be found in Figure 5.11 (b). In both cases, our method achieves notably better performance than Hist and UPOSI, while being comparable to Robust.

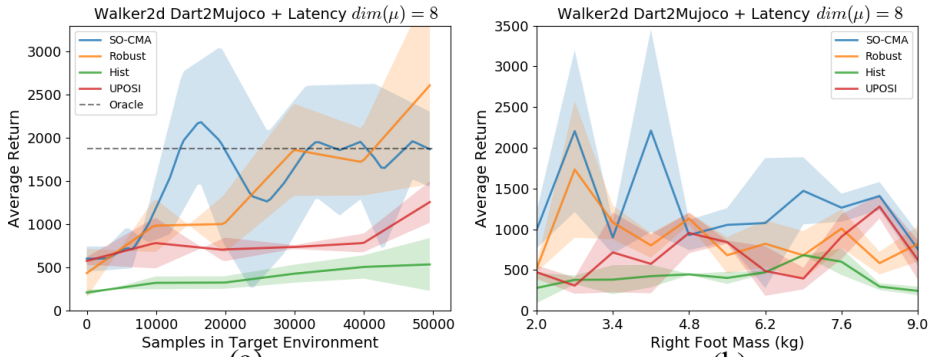


Figure 5.11: Transfer performance for the Walker2d example. (a) Transfer performance vs sample number in target environment on flat surface. (b) Transfer performance vs foot mass, trained with $30,000$ samples in the target environment.

5.3.7 HalfCheetah DART to MuJoCo with delay

In the third example, we train policies for the HalfCheetah environment from OpenAI Gym. We again test the performance of transfer from DART to MuJoCo for this example. In addition, we add a latency of 50ms to the target environment. We randomize 11 dynamic parameters in the source environment consisting of the mass of all body parts, the friction coefficient and the restitution coefficient during training, i.e. $dim(\mu) = 11$. The results

of the performance with respect to sample numbers in target environment can be found in Figure 5.12 (a). We in addition evaluate transfer to environments where the slope of the ground varies, as shown in Figure 5.12 (b). We can see that SO-CMA outperforms Robust and Hist, while achieves similar performance as UPOSI.

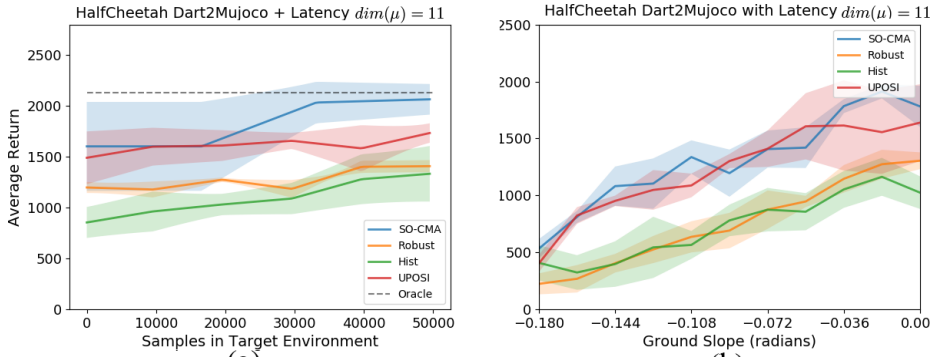


Figure 5.12: Transfer performance for the HalfCheetah example. (a) Transfer performance vs sample number in target environment on flat surface. (b) Transfer performance vs surface slope, trained with $30,000$ samples in the target environment.

5.3.8 Quadruped robot with actuator modeling error

As demonstrated by Tan et al. [58], when a robust policy is used, having an accurate actuator model is important to the successful transfer of policy from simulation to real-world for a quadruped robot, Minitaur (Figure 5.8 (d)). Specifically, they found that when a linear torque-current relation is assumed in the actuator dynamics in the simulation, the policy learned in simulation transfers poorly to the real hardware. When the actuator dynamics is modeled more accurately, in their case using a non-linear torque-current relation, the transfer performance were notably improved.

In our experiment, we investigate whether SO-CMA is able to overcome the error in actuator models. We use the same simulation environment from Tan et al. [58], which is simulated in Bullet [7]. During the training of the policy, we use a linear torque-current relation for the actuator model, and we transfer the learned policy to an environment with the more accurate non-linear torque-current relation. We use the same 25 dynamic parameters and corresponding ranges used by Tan et al. [58] for dynamics randomization during

training. When applying the robust policy to the accurate actuator model, we observe that the quadruped tends to sink to the ground, similar to what was observed by Tan et al. [58]. SO-CMA, on the other hand, can successfully transfer a policy trained with a crude actuator model to an environment with more realistic actuators(Figure 5.13 (a)).

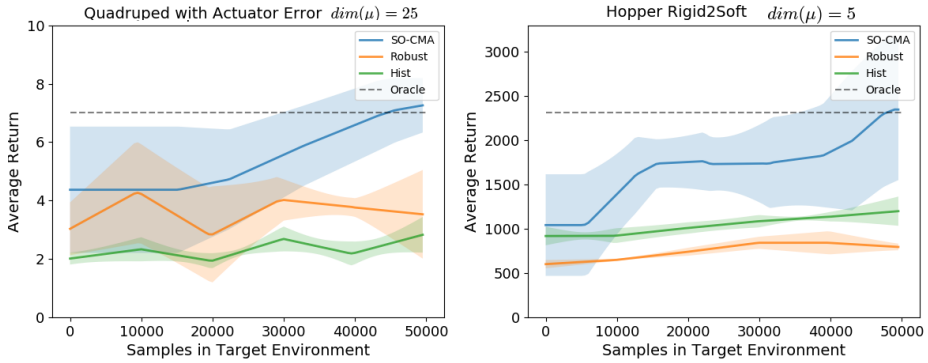


Figure 5.13: Transfer performance for the Quadruped example (a) and the Soft-foot Hopper example (b).

5.3.9 Hopper rigid to deformable foot

Applying deep reinforcement learning to environments with deformable objects can be computationally inefficient [121]. Being able to transfer a policy trained in a purely rigid-body environment to an environment containing deformable objects can greatly improve the efficiency of learning. In our last example, we transfer a policy trained for the Hopper example with rigid objects only to a Hopper model with a deformable foot (Figre 5.8 (e)). The soft foot is modeled using the soft shape in DART, which uses an approximate but relatively efficient way of modeling deformable objects [129]. We train policies in the rigid Hopper environment and randomize the same set of dynamic parameters as in the in the DART-to-MuJoCo transfer example with $\dim(\mu) = 5$. We then transfer the learned policy to the soft Hopper environment where the Hopper's foot is deformable. The results can be found in Figure 5.13 (b). SO-CMA is able to successfully control the robot to move forward without falling, while the baseline methods fail to do so.

5.3.10 Discussions

We have demonstrated that our method, SO-UP, can successfully transfer policies trained in one environment to a notably different one with a relatively low amount of samples. One advantage of SO-UP, compared to the baselines, is that it works consistently well across different examples, while none of the baseline methods achieve successful transfer for all the examples.

We hypothesize that the large variance in the performance of the baseline methods is due to their sensitivity to the type of task being tested. For example, if there exists a robust controller that works for a large range of different dynamic parameters μ in the task, such as a bipedal running motion in the Walker2d example, training a Robust policy may achieve good performance in transfer. However, when the optimal controller is more sensitive to μ, Robust policies may learn to use overly-conservative strategies, leading to sub-optimal performance (e.g. in HalfCheetah) or fail to perform the task (e.g. in Hopper). On the other hand, if the target environment is not significantly different from the training environments, UPOSI may achieve good performance, as in HalfCheetah. However, as the reality gap becomes larger, the system identification model in UPOSI may fail to produce good estimates and result in non-optimal actions. Furthermore, Hist did not achieve successful transfer in any of the examples, possibly due to two reasons: 1) it shares similar limitation to UPOSI when the reality gap is large and 2) it is in general more difficult to train Hist due to the larger input space, so that with a limited sample budget it is challenging to fine-tune Hist effectively.

We also note that although in some examples certain baseline method may achieve successful transfer, the fine-tuning process of these methods relies on having a dense reward signal. In practice, one may only have access to a sparse reward signal in the target environment, e.g. distance traveled before falling to the ground. Our method, using an evolutionary algorithm (CMA), naturally handles sparse rewards and thus the performance gap between our method (SO-CMA) and the baseline methods will likely be large if a sparse reward is

used.

5.4 Sim-to-Real Transfer for Biped Locomotion with Strategy Optimization

5.4.1 Overview

In this section, we describe a system to develop a biped locomotion controller by training a policy in simulation and deploying it on a consumer-grade robotic hardware (e.g. Darwin OP2 which costs less than $10,000 USD). Biped locomotion is a challenging task that requires precise control due to its inherent instability. Our approach split the controller synthesis process into three steps: Pre-training System Identification (pre-sysID), Policy Learning, and Policy Transfer. During pre-sysID, we command the real robot to perform a few basic movements such as standing up from a squatting pose, and collect the trajectory data for identifying the simulation parameters. Since the real-world data collected prior to the policy training may not be relevant to the task, the goal of pre-sysID is not to accurately identify the true value of model parameters, but only to approximate the range of model parameters in order to train a policy later. During the policy learning step, we simultaneously train a network that projects the model parameters to a low-dimensional latent variable, together with a family of policies that are conditioned on the latent space. We call the resulting policy Projected Universal Policy (PUP). The behavior of PUP can be modulated by a low-dimensional latent variable to adapt to different environments during the policy transfer step. After we have trained a PUP in the simulation, we perform Strategy Optimization (SO) on the real robot. Different from our previous algorithm, SO searches in the latent space of PUP, instead of the model parameter space $\boldsymbol{\mu}$.

We demonstrate our algorithm on training locomotion controllers for the Darwin OP2 robot to perform forward, backward and sideway walks. Our algorithm can successfully transfer the policy trained in simulation to the hardware in 25 real-world trials. We also evaluate the algorithm by comparing our method to two baseline methods: 1) identify a single model during system identification and train a policy for that model, and 2) use the

range of parameters from pre-sysID to train a robust policy.

5.4.2 Pre-training System Identification

The goal of a standard system identification procedure is to tune the model parameters $\boldsymbol{\mu}$ (e.g. friction, center of mass) such that the trajectories predicted by the model closely match those acquired from the real-world. One important decision in this procedure is the choice of data to collect from the real world. Ideally, we would like to collect the trajectories relevant to the task of interest. For biped locomotion, however, it is challenging to script successful locomotion trajectories prior to the policy training. Without task-relevant trajectories, any other choice of data can become a source of bias that may impact the resulting model parameters $\boldsymbol{\mu}$. Our solution to this problem is that, instead of solving for the optimal set of model parameters, Pre-sysID only attempts to approximate a reasonable range of model parameters for the purpose of domain randomization during policy learning. As such, we can use less task-relevant trajectories to cover a wide range of robot behaviors that may be remotely related to the task. In the case of locomotion, we use two set of trajectories for system identification: joint exercise without contact and standing/falling with ground contact. We use a set of pre-scripted actions to create these trajectories. (See details in Section 5.4.4).

Optimizing Range of Model Parameters

We optimized the model parameters $\boldsymbol{\mu}$ by creating simulated trajectories using the same pre-scripted actions that were used to collect the real-world trajectories. The fitness of a given $\boldsymbol{\mu}$ is given by the deviation between these simulated and real-world trajectories. Instead of trying to find a single simulation model that perfectly explains all the training data, we optimize for a set of models simultaneously, each of which fits a subset of the training trajectories. Specifically, we first use the entire set of trajectories to optimize a nominal set of model parameters $\hat{\boldsymbol{\mu}}$. We then select random subsets of the training trajectories, for

each subset we optimize the model parameters again with $\hat{\boldsymbol{\mu}}$ as initial guess. During the optimization of each subset, we add a regularization term $w_{reg}||\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}||^2$ to the objective function so that $\boldsymbol{\mu}$ will not go to local minima that are far away. We use $w_{reg} = 0.05$ in our experiments. This results in a set of optimized simulators, each of which can better reproduce a subset of the training data than $\hat{\boldsymbol{\mu}}$. We then extract the range of the simulation parameters by taking the element-wise maximum and minimum of the optimized $\boldsymbol{\mu}$'s and expand them by 10% to obtain the bounds $\boldsymbol{\mu}_{lb}$ and $\boldsymbol{\mu}_{ub}$.

We use CMA-ES [80], a sampling-based optimization algorithm, to optimize $\boldsymbol{\mu}$. To evaluate the fitness of a sampled $\boldsymbol{\mu}$, we compare the trajectories generated by the simulation to those from the hardware:

$$\begin{aligned} L = & \frac{1}{|D|} \sum_{D} \sum_{t} \|\bar{\mathbf{q}}_t - \mathbf{q}_t\| \\ & + \frac{10}{|D_{s,f}|} \sum_{D_{s,f}} \sum_{t} \|\bar{\mathbf{g}}_t - \mathbf{g}_t\| \\ & + \frac{20}{|D_s|} \sum_{D_s} \sum_{t} \|\Delta C_t^{feet}\|, \end{aligned} \tag{5.3}$$

where D denotes the entire set of input training trajectories from hardware, $D_{s,f}$ denotes the subset of standing or falling trajectories, and D_s contains only the standing trajectories. The first term measures the difference in the simulated motor position $\mathbf{q}$ and the real one $\bar{\mathbf{q}}$. The second term measures the difference in the roll and pitch of the robot torso between the simulated one $\mathbf{g}$ and the real one $\bar{\mathbf{g}}$. The third term measures the movement of the feet in simulation since the foot movement in the real trajectories is zero for those in D_s.

Neural Network PD Actuator

We model the biped robot as an articulated rigid body system with actuators at joints. For such a complex dynamic system, there are often too many model parameters to identify using limited amounts of real-world data. Among all the model parameters in the system,

Algorithm 6 System Identification of Parameter Bounds

Collect trajectories on hardware and store in D
$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} L(D, \boldsymbol{\mu})$
for $i = 1 : N$ **do**
    $D_i \leftarrow$ random subset of D
    $\boldsymbol{\mu}_i = \arg\min_{\boldsymbol{\mu}} L(D_i, \boldsymbol{\mu}) + w_{reg}\|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}\|^2$
$\boldsymbol{\mu}_{max}, \boldsymbol{\mu}_{min} \leftarrow$ per-dimension max and min of $\boldsymbol{\mu}_i$
$\boldsymbol{\mu}_{lb} = \boldsymbol{\mu}_{min} - 0.1(\boldsymbol{\mu}_{max} - \boldsymbol{\mu}_{min})$
$\boldsymbol{\mu}_{ub} = \boldsymbol{\mu}_{max} + 0.1(\boldsymbol{\mu}_{max} - \boldsymbol{\mu}_{min})$
return $\boldsymbol{\mu}_{lb}, \boldsymbol{\mu}_{ub}$

we found that the actuator is the main source of modeling error, comparing to other factors such as mass, dimensions, and joint parameters, similar to the findings in [59]. Therefore, we augment the conventional PD-based actuator model with a neural network to increase the expressiveness of the model, which we name Neural Network PD Actuator (NN-PD). For each motor on the robot, the neural network model takes as input the difference between the target position $\bar{\theta}_t$ and the current position of the motor θ_t, denoted as $\Delta\theta_t$, as well as the velocity of the motor $\dot{\theta}_t$, and outputs the proportional and derivative gains k_p and k_d. Unlike the high-end actuators used in [59], the actuators on Darwin OP2 are not capable of accurately measuring the actual torque being applied. As a result, we cannot effectively train a large neural network that outputs the actual torque. In our examples, we use a neural network model of one hidden layer with five nodes using tanh activation, shared across all motors. This results in network weights of 27 dimensions, which is denoted by $\boldsymbol{\phi} \in \mathbb{R}^{27}$.

We further modulate the differences among motors by grouping them based on their locations on the robot: $g \in \{\text{HEAD, ARM, HIP, KNEE, ANKLE}\}$. The final torque applied to the motor is calculated as:

$$\tau = clip(\rho_g k_p(\boldsymbol{\phi})\Delta\theta - \sigma_g k_d(\boldsymbol{\phi})\dot{\theta}, -\tilde{\tau}, \tilde{\tau}),$$

where the function $clip(x, b, u)$ returns the upper bound u or the lower bound b if x exceeds $[b, u]$. Otherwise, it simply returns x. We define learnable scaling factors ρ_g and σ_g for each

group, as well as a learnable torque limit $\tilde{\tau}$.

In addition to ϕ, σ_g and $\tilde{\tau}$, our method also identifies the friction coefficient between the ground and the feet and the center of mass of the robot torso. Identifying the friction coefficient is necessary because the surface in the real world can be quite different from the default surface material in the simulator. We found that the CAD model of Darwin OP2 provided by the manufacturer has reasonably accurate inertial properties at each part, except for the torso where the on-board PC, sub-controller and 5 motors reside. Thus, we include the local center of mass of the torso as an additional model parameter to identify.

The nominal model parameters $\hat{\boldsymbol{\mu}}$ we identify during pre-sysID include all the aforementioned parameters that has in total 41 dimensions. However, we fix the motor neural network weights ϕ and do not optimize the bounds for them. This is because neural network weights are trained to depend on each other and randomizing them independently might lead to undesired behavior. This results in the optimized parameter bounds $\boldsymbol{\mu}_{lb}, \boldsymbol{\mu}_{ub}$ to have dimension of 14.

5.4.3 Training Projected Universal Policy

In the previous section (5.3), we describe training of a universal policy (UP) $\pi_{up} : (\mathbf{o}, \boldsymbol{\mu}) \mapsto \mathbf{a}$ explicitly conditioned on the model parameters $\boldsymbol{\mu}$. By performing strategy optimization (SO) to find the optimal $\boldsymbol{\mu}$ that achieves best performance in the target environment, we are able to achieve successful sim-to-sim transfer. However, the required sample number (50,000) is too high to be applied on a biped robot.

To reduce the sample number during transfer, we exploit the redundancy in the space of $\boldsymbol{\mu}$ in terms of its impact on the policy. For example, increasing the mass of a limb will cause a similar effect on the optimal policy to increasing the torque limit of the motor connected to it. Therefore, we learn a projection model that maps $\boldsymbol{\mu}$ down to a lower-dimensional latent variable $\mathbf{c} \in \mathbb{R}^M$, where M is the dimension of the latent space ($M = 3$ in our experiments). We then condition the control policy directly on $\mathbf{c}$, instead of $\boldsymbol{\mu}$. We connect

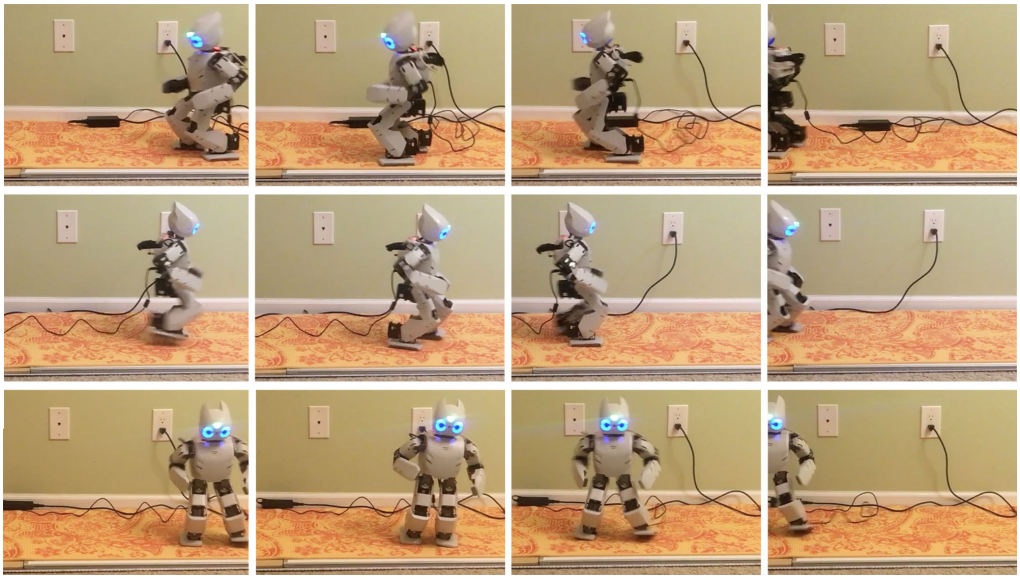


Figure 5.14: Illustration of locomotion policies deployed on the Darwin OP2 robot. Top: walk forward. Middle: walk backward. Bottom: walk sideways.

the last layer of projection module to the policy's input layer via a tanh activation such that the weights of both the projection module and the policy are trained together using the policy learning algorithm PPO [56]. This results in a policy that can exhibit different behaviors, modulated by $\mathbf{c}$. We call this policy a Projected Universal Policy (PUP): π_{pup} : $(\mathbf{o}, \mathbf{c}) \mapsto \mathbf{a}$.

5.4.4 Results

Experiment Setup

We test our algorithm on the Robotis Darwin OP2 robot. Darwin OP2 has 20 Dynamixel MX-28T servo motors in total, 2 on the head, 6 on the arms and 12 on the legs, all of which are controlled using target positions through a PID controller. We set the P gain, I gain and D gain of all the motors to be 32, 0 and 16 on the hardware. Note that the PID controller for the actual motor is defined at the pulse width modulation (PWM) level, while the PD controller used in the simulation is defined at the torque level. Thus the gains used on the hardware is not transferable to the simulation. MX-28T provides decent sensing accuracy for the position and velocity of the motor. However, reading the position

or velocity from all the motors takes about 10ms, which limits our control frequency when both data are used. In this work, we instead use only the positions from the motors, and provide two consecutive motor position readings to the policy to provide information about the velocities. Darwin OP2 is also equipped with an on-board IMU sensor that provides raw measurements of angular velocity and linear acceleration of the robot torso. In order to have good estimation of the orientation or the robot, we need to collect and integrate data from the IMU sensor at high frequency. However, this is not possible because the IMU and the motors share the communication port. Therefore, instead of the on-board IMU, we use the Bosch bno055 IMU sensor for estimating the orientation of the robot. With this augmentation, we can reach a control frequency of $33Hz$. We use a physics-based simulator, Dart [5], to simulate the robot's behavior under different control signals.

To evaluate our approach, we train locomotion policies that control the Darwin OP2 robot to walk forward, backward and sideways in simulation and transfer to the real hardware. We first collect motion trajectories of the real robot performing manual-scripted movements. For each motor on the robot, we apply a step function action starting from a random pose and record their responses while suspending the robot to avoid ground contact. One such example can be seen in Figure 5.18. We use step functions of magnitude 0.1, 0.3 and 0.6 to collect motor behaviors at different speeds. In addition, we also design trajectories where the robot stands up and falls in different directions. For trajectories that involve ground contact, we also record the estimated orientation from the IMU sensor. The set of movements we use can be seen in the supplementary video [1].

The robot is tasked to walk on a yoga mat that lies on top of a white board, as shown in Figure 5.14. We choose this deformable surface to better provide protection for the robot. The performance of the policy can be viewed in the supplementary video.

For all examples in this section, the observation space includes the position of motors and the estimated orientation represented in Euler angles for two consecutive timesteps.

[1] https://youtu.be/bq8xZgbLHcw

The action space is defined as the target positions for each motor. To accelerate the learning process in simulation, we use a reference trajectory of robot stepping in place that was generated manually. The action of the policy is to apply adjustment to the reference trajectory:

$$\mathbf{q}_{target} = \mathbf{q}_{ref} + \delta\pi_{pup}(\mathbf{s}; \mathbf{c}^*),$$

where δ controls the magnitude of the adjustment and the policy π_{pup} outputs a value in $[-1, 1]$. We use $\delta = 0.3$ for walking forward and sideways and $\delta = 0.2$ for walking backwards. Note that the reference trajectory does not need to be dynamically feasible, as tracking our stepping-in-place reference trajectory causes the robot to fall immediately. Similar to [61], we also discretize the action space into 11 bins in each dimension to further accelerate the policy training.

We use Proximal Policy Optimization (PPO) to train the control policies and use the following reward function:

$$\mathcal{R}(\mathbf{s}, \mathbf{a}) = w_v E_v + w_a E_a + w_w E_w + w_t E_t + E_c.$$

The first term $E_v = \|\dot{x}\|$ encourages the robot to move as fast as possible in the direction $\mathbf{x}$. The second and third term $E_a = ||\boldsymbol{\tau}||^2$, $E_w = \boldsymbol{\tau} \cdot \dot{\mathbf{q}}$ penalize the torque and work applied to the motors, where $\boldsymbol{\tau}$ is the resulting torque applied to the motor under the action $\mathbf{a}$. $E_t = ||\mathbf{q}^t - \mathbf{q}^t_{ref}||^2$ rewards the robot to track the reference trajectory, where $\mathbf{q}^t_{ref}$ denotes the reference trajectory at timestep t. Finally, $E_c = 5$ is a constant reward for not falling to the ground. We use an identical reward function with $w_v = 10.0, w_a = 0.01$, $w_w = 0.005$, $w_t = 0.2$ for all of the presented examples. We also use the mirror symmetry loss proposed in [11] during training of PUP, which we found to improve the quality of the learned locomotion gaits. For controlling the robot to walk in different directions, we rotate the robot's coordinate frame such that the desired walking direction is aligned with the positive x-axis in the robot frame.

Baselines

We evaluate our method by comparing it with two baselines. For the first baseline, we optimize for a single model $\boldsymbol{\mu}$ during Pre-sysID instead of a range of $\boldsymbol{\mu}$, and use the model to train a policy. We denote this baseline "Nominal". The second baseline uses the range of $\boldsymbol{\mu}$ computed by Pre-sysID and trains a robust policy through domain randomization with that range. We denote the second baseline "Robust".

To account for uncertainty in the sensors and networking, we model additional noise in the simulation during policy training for our method and the two baselines. Specifically, we randomly set the control frequency to be in $[25, 33]$Hz for each rollout, add a bias to the estimated orientation drawn from $\mathcal{U}(-0.3, 0.3)$ and add a Gaussian noise of standard deviation 0.01 to the observed motor position. In addition, we add noise drawn from $\mathcal{U}(-0.25, 0.25)$ to the input $\boldsymbol{\mu}$ during the training of PUP to improve the robustness of the policy.

Performance on Locomotion Tasks

Figure 5.15 and Figure 5.16 show the comparison between our method and the baselines. We evaluate a trained policy by running it 5 times on the real hardware, and measure the distance and time before the robot loses balance or reaches the end of the power cable. Our method clearly outperforms the baselines and is the only method that can control the robot to walk to, and occasionally beyond, the edge of the white board (at 0.8m). Because PUP is trained to be specialized for different environments, it learns to take larger steps than Robust, which tends to take conservative actions. This results in a faster walking gait, and may also have contributed to the larger variance seen in our policy. An illustration of the three locomotion tasks with our trained policies can be seen in Figure 5.14.

Effect of using NN-PD Actuators

Our method models the motor dynamics as a neural network paired with a PD controller. Here we examine the necessity of having this additional components in the model. Specif-

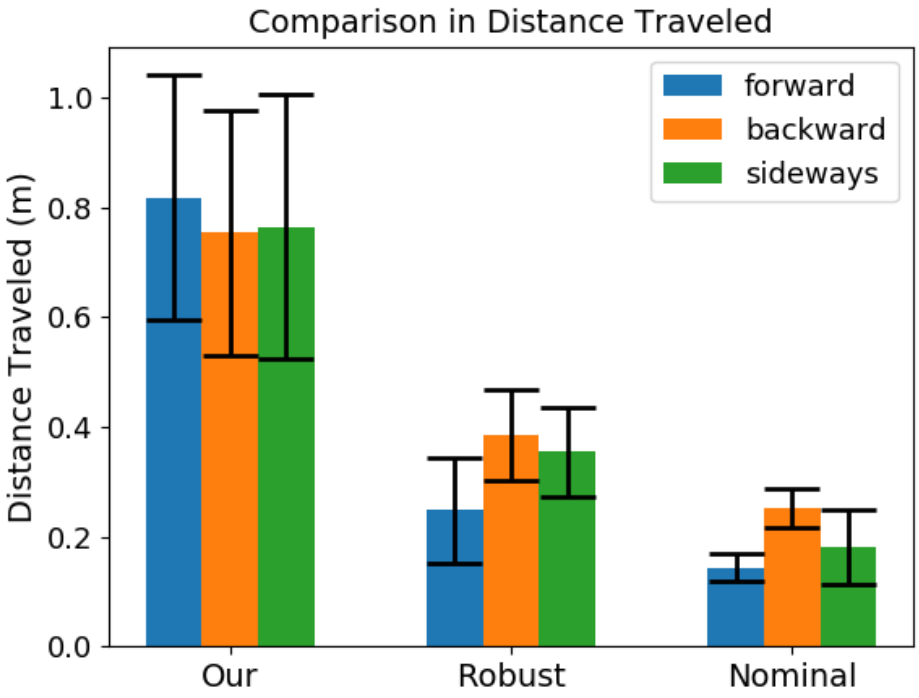


Figure 5.15: Comparison of the distance travelled by the robot using our method and the baselines. Error bars indicate one standard deviation from five runs of the same policy.

ically, we identify two models, one with NN-PD controllers and one without (PD only), using the same set of real-world data. Figure 5.17 shows the the optimization curve over 500 iterations for both models. We can see that NN-PD is able to achieve a notably better loss compared to PD only. To further demonstrate the behavior of the two models, we plot the simulated motor position for the hip joint when a step function is applied, and the estimated pitch of the torso when a trajectory controls the robot to fall forward, as shown in Figure 5.18 (a) and (b). We can see that NN-PD is able to better reproduce the overall behavior than PD only.

Identified Model Parameter Bounds

Figure 5.19 visualizes $\boldsymbol{\mu}_{lb}$ and $\boldsymbol{\mu}_{ub}$ identified by the pre-sysID stage. We normalize the search range of each model parameter to be in $[0, 1]$ and show the identified bounds as the blue bars. The red lines indicate the nominal parameters $\hat{\boldsymbol{\mu}}$ optimized using the entire set of

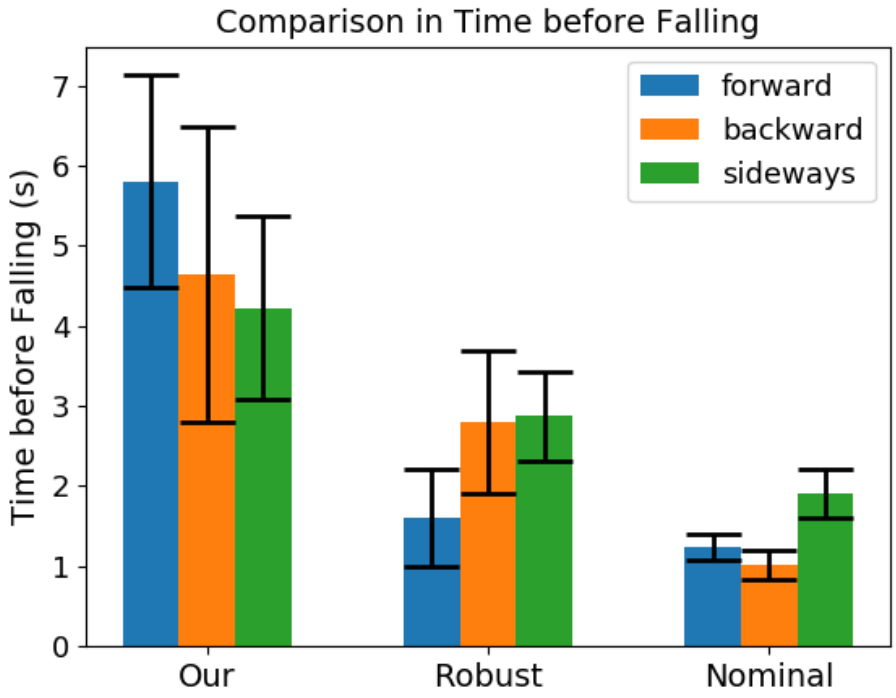


Figure 5.16: Comparison of the elapsed time before the robot loses balance using our method and baselines. Error bars indicate one standard deviation from five runs of the same policy.

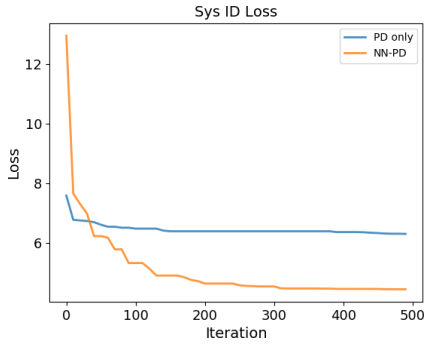


Figure 5.17: Comparison of system identification performance with and without NN-PD actuators. Both models are optimized using the same set of real-world data and the reported loss is calculated according to Equation 5.3.

pre-sysID trajectories. Some parameters, such as σ_{ankle}, have tighter bounds, which indicate higher confidence in the optimized values for those parameters. The parameters with wider range indicate that no single value of $\boldsymbol{\mu}$ can explain all the training trajectories well and naively using the nominal values for these parameters may lead to poor transfer perfor-

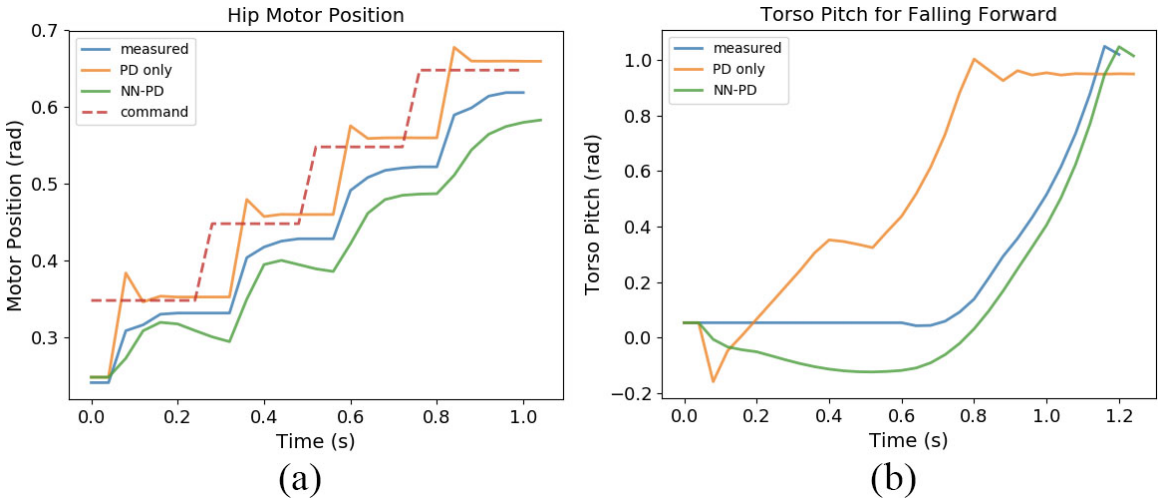


Figure 5.18: System identification performance comparison of NN-PD and PD only on (a) the hip motor position during step function command with magnitude 0.1 and (b) torso pitch during falling forward motion.

mance. One example of such parameters is the bounds for $\tilde{\tau}$. Upon further examination, we found that the identified $\tilde{\tau}$ tends to be bipolar depending on whether the subset of training trajectories involves contact or not. These phenomena suggest that our current model is still not expressive enough to explain all the real-world observations and further improvement in modeling may be necessary for transferring more challenging tasks. We also note that, partially due to the wide range of motions for pre-sysID, the identified bounds are not necessarily useful for the tasks of interest. For example, the parameters associated with the head, ρ_{head} and σ_{head}, have wide bounds but their impact to locomotion tasks is relatively small. During the training of PUP, the projection module will learn to ignore the variations in these parameters.

5.4.5 Discussions

We have demonstrated a learning system based on deep reinforcement learning and strategy optimization for learning robotic controllers in physics simulation and applying them to the real hardware. The key idea is to train a Projected Universal Policy, whose behavior is modulated by a low dimensional latent variable, and then search for the latent variable on

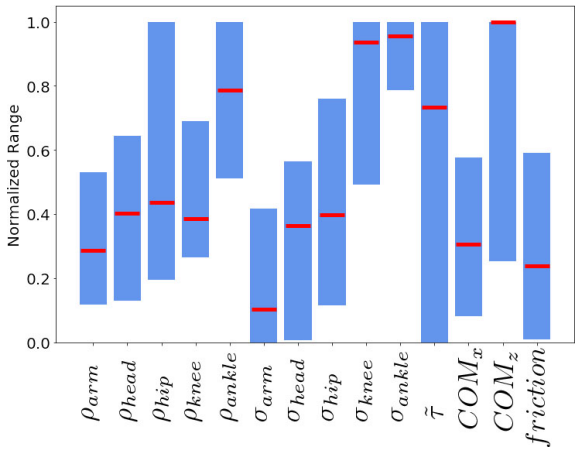


Figure 5.19: Identified model parameter bounds (blue bars) and the nominal parameters (red lines).

the real robot for the best performance. We demonstrate our method on training locomotion policies for the Darwin OP2 robot to walk forward, backward and sideways and transfer to the real robot using 25 trials on the hardware.

During strategy optimization, our method uses additional task-relevant data to help identify the optimal conditioned policy $\pi_{pup}(\mathbf{s}; \mathbf{c}^*)$. To provide a fair comparison, one could also use task-relevant data to further improve the baseline policies. However, to fully take advantage of these trajectories for policy learning, one would need to upgrade sensor instrumentation for measuring global position and orientation needed in reward function evaluation. In contrast, our method only uses these trajectories for transfer with very simple fitness function that only measures the traveling distance and the elapsed time. In addition, the size of the task-relevant data (less than 2500 steps) is only enough to perform one iteration of PPO in a typical setting. In comparison, we use $20,000$ steps per learning iteration in simulation. For those reasons, we do not believe that such a small amount of task-relevant data can further improve the results of baseline methods in our experiments.

The set of model parameters $\boldsymbol{\mu}$ used in our work is currently chosen manually based on prior knowledge about the robot and the tasks of interest. Although it works for Darwin OP2 learning locomotion tasks, it is not clear as to how well it can be applied to different robotic hardware or different tasks, such as picking up objects or climbing ladders. Investi-

gating a systematic and automatic way to select model parameters would be an interesting future research direction.

5.5 Meta Strategy Optimization

5.5.1 Overview

As shown in the previous section, strategy optimization with projected universal policy (SO-PUP) has demonstrated successful sim-to-real transfer for biped locomotion problems. However, during the training of SO-PUP, the latent space of context variables are acquired through the projection network that has never experienced the adaptation process before. As a result, the projected universal policy may not learn a latent space that is not in favor of fast adaption to the real environments.

In this section, we develop a transfer learning algorithm that extends the idea of SO-PUP by exposing the learning agent to the same strategy optimization process during both training and testing phases. This meta-training allows the agents to learn a better latent policy space that is suitable for fast adaptation to new situations. As such, we name our new method Meta Strategy Optimization (MSO).

We demonstrate our proposed algorithm on training locomotion policies for the Ghost Robotics Minitaur [130], a quadruped robot. Our algorithm can successfully train locomotion policies that can be applied to the real hardware by adjusting its simulation-acquired behavior. In addition, we design two adaptation tasks for the real robot, walking with a weakened leg and climbing a slope, and a set of additional tasks in a simulated environment. We show that MSO is extremely data efficient (≤ 15 rollouts or 75 seconds of data) to adapt the policies to novel situations in the target environment. We compare our method to two baseline methods: domain randomization [58] and strategy optimization with a projected universal policy [14]. Our results show that MSO outperforms both baselines in the simulated and the real environments.

5.5.2 Meta Strategy Optimization

The key idea behind MSO is that we adopt the same adaptation process to obtain the latent input to the policy during both training and testing. Therefore, our policy directly takes latent variables $\mathbf{c}$ as inputs.

We solve the following optimization problem during training in simulation:

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\boldsymbol{\mu}}[\max_{\boldsymbol{c}} J_{\boldsymbol{\mu}}(\boldsymbol{c}, \theta)], \tag{5.4}$$

where θ is the weight of the policy network, $J_{\boldsymbol{\mu}}(\boldsymbol{c})$ is the performance of the strategy $\pi_\theta(\mathbf{o}, \boldsymbol{c})$ when the physics parameters are $\boldsymbol{\mu}$. Note that we refer $\boldsymbol{\mu}$ to the physics parameters for clarity and consistency to previous works. However, one can easily extend it to include parameters from other components of the MDP such as the reward function.

Directly solving Equation 5.4 is challenging for two reasons. First, the objective term involves strategy optimization inside the expectation, which makes it difficult to compute the gradient with respect to the policy parameters θ. Second, every single evaluation of the policy parameters θ involves performing SO to get the optimal strategy (Equation 5.6), which increases the computational cost significantly.

We propose a practical algorithm by observing that the optimization problem in Equa-

Algorithm 7 Meta Strategy Optimization

```
Randomly initialize policy weights θ_1.
for t = 1 : k do
    Sample n tasks {μ_i | i = 1, ..., n}.
    For each μ_i, solve Eq. 5.6 with θ_t and obtain c_{μ_i,t}.
    for j = 1 : h do
        Randomly sample a pair of (c_{μ,t}, μ).
        Collect rollouts with p_μ and π_{θ_t}(o, c_{μ,t}).
        Obtain θ_{t+1} by solving Equation 5.7.
return π_{θ_k}
```

tion 5.4 can be written as:

$$\theta^*, \boldsymbol{c}(\boldsymbol{\mu})^* = \arg\max_{\theta, \boldsymbol{c}(\boldsymbol{\mu})} \mathbb{E}_{\boldsymbol{\mu}}[J_{\boldsymbol{\mu}}(\boldsymbol{c}(\boldsymbol{\mu}), \theta)], \tag{5.5}$$

where $\boldsymbol{c}(\boldsymbol{\mu})$ is a mapping from the tasks $\boldsymbol{\mu}$ to its corresponding latent variable. We can then solve the optimization problem in an approach similar to Coordinate Descent [131], where we alternate between optimizing the latent variables $\boldsymbol{c}(\boldsymbol{\mu})$ and the policy network parameters θ:

$$\boldsymbol{c}_{\boldsymbol{\mu},t} = \arg\max_{\boldsymbol{c}} J_{\boldsymbol{\mu}}(\boldsymbol{c}, \theta_t) \tag{5.6}$$

$$\theta_{t+1} = \arg\max_{\theta} \mathbb{E}_{\boldsymbol{\mu}}[J_{\boldsymbol{\mu}}(\boldsymbol{c}_{\boldsymbol{\mu},t}, \theta)], \tag{5.7}$$

where t is the iteration number.

Algorithm 7 describes the MSO algorithm in more details. For each iteration of policy learning, we first sample a set of n tasks from the simulator and perform strategy optimization to obtain the current best strategies for these tasks. We then perform h steps of policy updates with the fixed set of task-strategy pairs. In our experiments, we use $n = 5$ and $h = 30$.

By computing the latent variable $\mathbf{c}$ using strategy optimization, MSO avoids the need to compute a projection from $\boldsymbol{\mu}$ to $\mathbf{c}$ and thus can handle tasks with larger dimensions than SO-PUP. More importantly, by matching the process of obtaining the latent variable during training and testing, MSO can implicitly shape a latent space of control behaviors that is more suitable for strategy optimization when adapting to novel scenarios.

5.5.3 Results

We aim to answer the following questions in our experiments: 1) Does MSO achieve better performance than the baseline methods DR [58] and SO-PUP [14] in adapting to new

dynamics and rewards? 2) Does MSO train policies that can be successfully transferred to real robots and adapt to novel scenarios in the real world? 3) Is MSO sensitive to the specific choice of hyper-parameters? 4) Does MSO achieve better performance on adapting to new dynamics than gradient-based meta learning algorithms? To answer these questions, we design a set of experiments in both simulation and real-world. Videos of our results can be seen in the supplement video [2].

Experiment setup

We use Minitaur from Ghost Robotics [130] as the robot platform to evaluate our algorithm. Minitaur has eight direct-drive actuators, two on each leg. In this work, we use a Proportional-Derivative controller (P gain is 0.5 and D gain is 0.005) to track the desired motor positions, which is the output of the policy. Minitaur is equipped with motor encoders to read the motor angles and an IMU sensor to estimate the orientation and angular velocity of the robot body. The robot is controlled at a frequency of 50 Hz.

We build a physics simulation of the Minitaur in PyBullet [7], a Python module that extends the Bullet Physics Engine. Our simulator incorporates the actuator model [58], but we do not perform a thorough system identification for its parameters. As shown in our experiments, a naïve domain randomization technique does not give us a transferable policy directly.

The observation space of the robot consists of the current motor angles, the roll, pitch of the base, as well as their time derivatives. We design a reward function that encourages the robot to move forward:

$$\mathcal{R} = clip((\mathbf{p}_n - \mathbf{p}_{n-1}) \cdot \mathbf{d}/dt, -\bar{v}, \bar{v}), \tag{5.8}$$

where $\mathbf{p}_n$ denotes the position of the robot base at timestep n, $\mathbf{d}$ is the desired moving direction, dt is the control timestep, and $\bar{v}$ is a velocity threshold for safety reasons. We use

[2] Video available at: `https://www.youtube.com/watch?v=Mm3IIEZ0-Nw`

$dt = 0.02$s and $\bar{v} = 1$m/s in our experiments. Each episode of simulation has a maximum horizon of 250 steps (5s). The episode is terminated early if the robot falls, determined by the roll and pitch angles of the base.

We represent the locomotion policy using a feed-forward neural network with two hidden layers, each consists of 64 neurons. We use Augmented Random Search (ARS), a policy optimization algorithm, for training the locomotion policy in simulation [108]. In our experiments, we sample 92 perturbations for each iteration and use the top 23 perturbations to update the policy weights (see Chapter 3 for more details). Although ARS has only been demonstrated for training linear policies, we find it also effectively in training neural network policies. We choose ARS because it can better leverage large scale computational resource, though MSO can also be applied to other on-policy RL algorithms such as PPO [56]. We use Bayesian Optimization to perform SO and limit the maximum episode number to 25 during training.

We compare MSO to two baselines: domain randomization (DR) [58] and strategy optimization with projected universal policy (SO-PUP) [14]. We run ARS for 1500 iterations for all methods and we use a two-dimensional latent space for MSO and SO-PUP. Table 5.1 shows the physics parameters and their corresponding range we use during training. During our experiments on the hardware, we find that 15 episodes are sufficient to achieve successful adaptation. Thus we choose 15 episodes during testing for both MSO and SO-PUP. To reduce the influence of the stochastic learning process, we train five policies for each method. Each trained policy is then evaluated on $1,500$ sampled tasks from the designed task distributions for all simulated adaptation experiments (Section 5.5.3).

We further compare MSO to two gradient-based meta learning algorithms: Model-Agnostic Meta Learning (MAML) [71] and No-Reward Meta Learning (NoRML) [74] on a simulated Hopper robot[3]. During training of all methods, we vary the ground friction

[3]We use the implementation of MAML and NoRML from https://github.com/google-research/google-research/tree/master/norml. The Hopper environment was modeled and simulated using Dart [**dart2018**], and can be found here: https://github.com/DartEnv/dart-env.

in $[0.1, 1.0]$, and the weight of the torso in $[2, 15]kg$. During testing, we evaluate the performance of the policy with an extended range ($[0.1, 1.9]$ for ground friction and $[2, 28]kg$ for torso weight) to test the generalization performance. For MSO, we run ARS for 600 iterations, each with 32 perturbations. We use the top 8 perturbations for updating the policy. The rest of the hyper-parameters are the same as the ones in Minitaur experiments. For MAML and NoRML, we run the policy update for 1000 iterations and use the default hyper-parameters for the algorithms. We allow 25 episodes during adaptation for all three methods. The results can be found in Section 5.5.3.

Adaptation tasks

We design the following tasks on the real robot to evaluate the performance of MSO:

1) Sim-to-real transfer. The first task is to transfer the policy trained in simulation to the real Minitaur robot. Although we use the nonlinear actuator model from Tan et al. [58], the reality gap in our case is still large as we use a different version of Minitaur and we do not perform additional system identification.

2) Weakened motors. It is common for real robots to experience motor weakening, e.g. due to over heating. In this task, we test the ability of MSO to adapt to weakened motors by setting the P gain to 0.2 for the two motors on the front right leg of Minitaur. Such strength reduction (60%) is beyond the range that the policy has seen during training.

3) Climbing up a slope. In this task, we place the robot on a slope of about 10 degrees constructed by a white board and task the robot to climb up the hill. This is a challenging task because during training in the simulation the robot has only seen flat ground.

In addition, we design the following tasks in simulation for a more comprehensive analysis of the adaptation performance of MSO:

1) Extended randomization. In this task, we sample dynamics from the same set of parameters used in training (Table 5.1), but with an extended range that is $\sim 30\%$ wider. We also reject samples that lie within the training range to focus on generalization capability.

This gives us a large space of testing dynamics that have not been seen during training.

2) Climbing up slopes. We also evaluate MSO for climbing up a hill in simulated environments. We randomize the angle of the slope in $[5, 20]$ degrees during evaluation.

3) Motor offset. One of the common defects of actuators is that the zero position is wrong. We evaluate the ability of MSO to adapt to such issues in this task. Specifically, we add an offset sampled in $[-35, 35]$ degrees to the observed angles of the two motors on the front left leg.

4) Carrying an object. All tasks above involves adapting to changes in dynamics only. In this task, we design a scenario where both dynamics and reward changes. Specifically, we ask the robot to carry a box of 1 Kg while running forward. The new reward is how far the box is carried without falling to the ground. This task stresses the need of adapting the behavior of the policy, and a robust policy with a single behavior is unlikely to succeed.

Note that the testing variations in the simulated tasks (slope, motor offset, object) are not included during the training of the policy. The policy needs to adapt to this novel task by leveraging the latent space acquired for the diverse set of dynamics seen during training. For all simulated tasks except for extended randomization range, we also need to determine what values to use for the parameters randomized during training. As there is no single set of values that is representative of the robot, we also randomize these parameters using the same training range (Table 5.1) for those tasks.

Table 5.1: Randomized parameters and their range used in training.

parameter	**lower bound**	**upper bound**
mass	60%	160%
motor friction	0.0Nm	0.2Nm
inertia	25%	200%
motor strength	50%	150%
latency	0ms	80ms
battery voltage	10V	18V
contact friction	0.2	1.25
joint friction	0.0Nm	0.2Nm

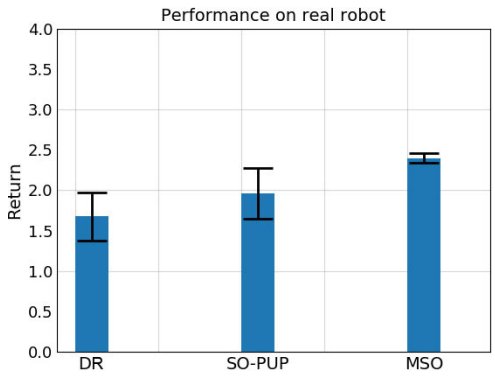


Figure 5.20: Sim-to-real performance comparison on the Minitaur robot (corresponding to Task 1: Sim-to-real transfer as described in 5.5.3). Error bar denotes one standard deviation.

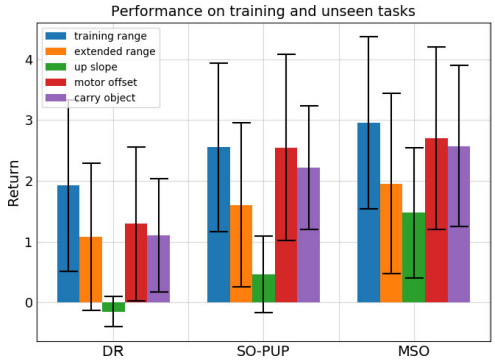


Figure 5.21: Comparison of performance on the training randomization range and generalization to unseen tasks. Error bar denotes one standard deviation.

Results on real robot

We evaluate MSO on real Minitaur robot for the three tasks described in Section 5.5.3. For MSO and the baseline methods, we use the policy with the highest training performance among the five trials to deploy on the real hardware. For MSO and SO-PUP, we allow 15 episodes for the adaptation and repeat the best policy for three times to obtain the final performance. For the sim-to-real task, we evaluate all three methods and report the result in Figure 5.20. We see that MSO is able to not only achieve a better performance on average, but also obtain lower variance in performance.

For the task of weakened motor and slope climbing, we compare MSO to DR and SO-PUP. As seen in the supplement video, when the front right leg is weakened, the robot lacks the strength to lift it up, and MSO finds a strategy that drags the front right leg forward

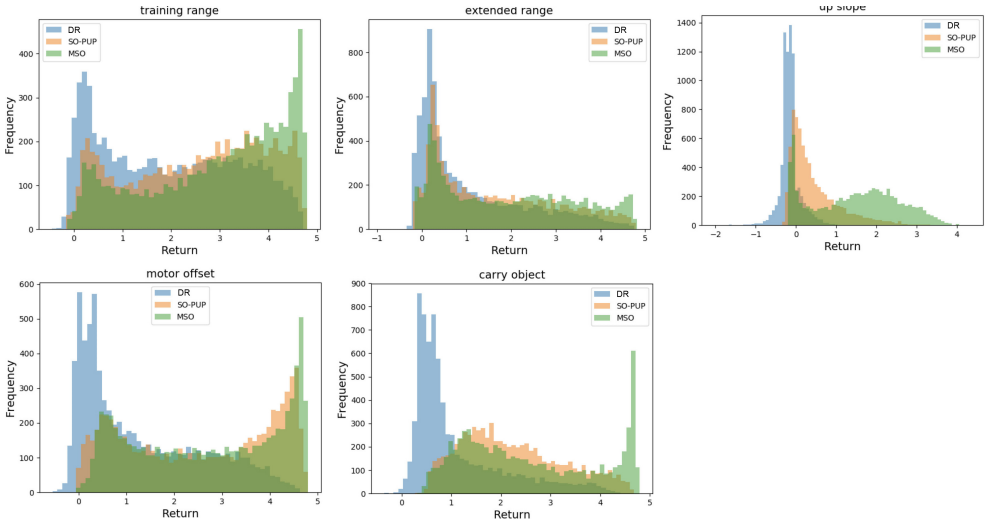


Figure 5.22: Histograms for the returns of the sampled tasks in different adaptation problem. Each method was evaluated on $7,500$ sampled tasks for each adaptation problem.

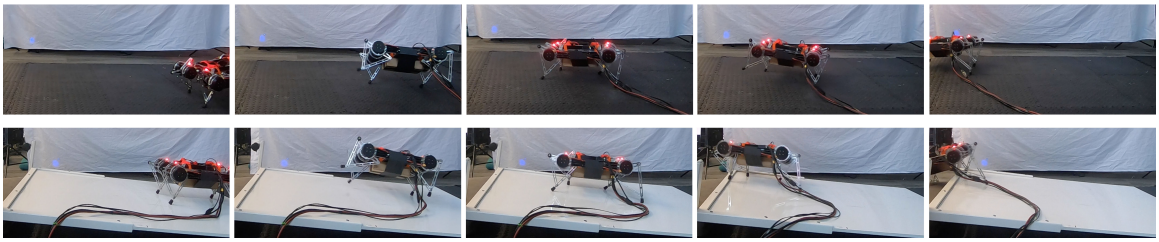


Figure 5.23: Policy trained by MSO adapts to new tasks: front right leg weakened (top), walking up a slope (bottom).

without falling. On the other hand, DR still assumes full strength of the front right leg and relies on it to lift the base of the robot up, leading to it losing balance. Similarly for the task of climbing up the hill, MSO is able to find a strategy that successfully take the robot up the hill and go beyond the slope, while DR leads to the robot falling backward as it has only seen flat ground. SO-PUP is able to learn different strategies that allow it to perform sim-to-real transfer to some extent, yet the resulting strategies are not rich enough to overcome the novel tasks.

More analysis in simulation

We evaluate our method in simulated adaptation tasks to provide a more comprehensive analysis of our algorithm. We evaluate the performance of MSO and the baseline methods by testing them on the dynamics within the training range, as well as on the four adaptation tasks described in Section 5.5.3: extended randomization, climbing up a slope, biased motor zero position, and carrying an object.

Figure 5.21 shows the mean and standard deviation for the three methods on different adaptation tasks. The statistics for each experiment are computed from $7,500$ samples. We also plot the histograms for the returns for each set of experiment to understand the reward distributions over a wide range of tasks (Figure 5.22). For all adaptation tasks, MSO is able to outperform both SO-PUP and DR. Notably, for the task of climbing up a slope, MSO achieved a clear advantage over the baseline methods, while DR is not able to achieve positive return. On the other hand, the difference between MSO and SO-PUP is smaller when an offset is added to the observed motor angle, while DR performs much worse. These results suggest that some tasks, such as climbing up the slope, are more sensitive to latent space qualities than other tasks. MSO also works well for the task of carrying the object, where the policy needs to adapt to changes in both dynamics and reward. As seen in the supplement video, MSO can successfully find a strategy that stabilizes the base of the robot to prevent the object from falling to the ground, while the baseline methods achieves

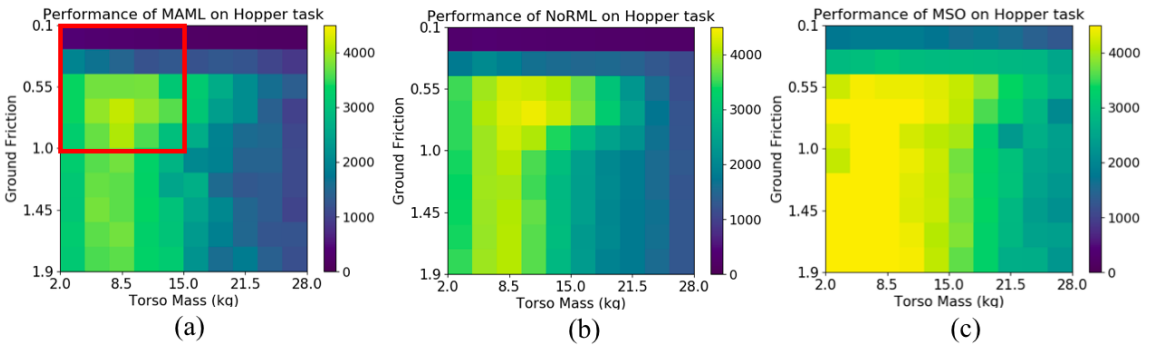


Figure 5.24: Performance comparison between MAML (a), NoRML (b), and MSO (c) on the Hopper task. The squared region in (a) denotes the range of the training dynamics for all three methods. The color in the plot represents the performance of a task. The better the method performs with the dynamics parameters setting, the lighter the grid color is.

worse performance.

Ablation study

We investigate how sensitive our algorithm is to different choices of hyper-parameters. In particular, we vary three key parameters for MSO: 1) e: the number of episodes allowed in SO during training, 2) l: the dimension of the latent space, and 3) h: the number of iterations between each SO during training. Our nominal model uses $e = 25$, $l = 2$ and $h = 30$ for the three parameters. We vary one parameter at a time from the nominal setting and pick two values for each parameter being ablated. We test all variations of MSO on the training performance and the extended randomization task with $7,500$ samples each. During testing, we allow 15 episodes for adaptation for all variations. Table 5.2 shows the

Table 5.2: Ablation study for the MSO algorithm.

parameters	**mean return (training)**	**mean return (extended)**
e=25, l=2, h=30	2.95	1.95
e=15	2.91	1.85
e=1	2.36	1.51
l=1	2.84	1.85
l=5	2.97	1.95
h=15	2.70	1.78
h=50	3.01	1.94

result of the ablation.

In general, our method is not very sensitive to different hyper-parameters. Interestingly, even when a single episode is allowed for SO during training, i.e. a random strategy is selected, the resulting policy can still outperform DR notably. This is possibly because training a policy in this setting is similar to training a set of DR policies with different random seeds, and during testing, the best performing one will be picked.

Comparison to gradient-based meta learning

Finally, we evaluate our method on a completely different domain, training a simulated Hopper robot to hop forward, to compare it against two gradient-based meta learning algorithms: MAML [71] and NoRML [74]. We evaluate all methods on dynamics that extend the training range by 100%, which are shown in Figure 5.24. We observe that MSO significantly outperforms the other two methods both in terms of the training and generalization performance. We believe this is due to that MSO optimizes the latent input $\mathbf{c}$ in a low dimensional space (2D in our case), which allows more sample-efficient adaptation than gradient-based adaptation methods that need to adjust the entire policy network.

5.5.4 Discussions

We have presented a learning algorithm for training locomotion policies that can quickly adapt to novel environments that are not seen during training time. The key idea to our method, Meta Strategy Optimization (MSO), is a meta-learning process that learns a latent strategy space suitable for fast adaptation during training, and quickly searches a good strategy to adapt to new rewards and dynamics during testing. We demonstrate MSO on a variety of simulated and real-world adaptation tasks, including walking on a slope, weakened motor, and carrying objects. MSO can successfully adapt to the novel tasks in 15 episodes and outperforms other baseline methods.

Though MSO can successfully transfer policies to environments that are notably dif-

ferent from the training environments, it assumes that the testing environment does not change significantly over time. This restricts the type of tasks that MSO can be applied to. For example, if the robot needs to walk across an slippery surface and a rough surface, it would require changing the strategy when the surface type changes. One possible future direction to address this issue is to adopt the idea of hierarchical RL [132, 133] by treating the MSO-trained policy as a lower-level policy and train a higher-level policy that outputs the strategy. This will also enable the policy to adapt in an online fashion.

CHAPTER 6

TRAINING SAFETY-AWARE LOCOMOTION CONTROLLER

6.1 Motivation

Humans have the natural ability to assess the risk of the situation and adjust their actions to incorporate the estimated risk: we will walk slowly if we are holding a cup of hot coffee that is nearly full in fear of spilling, while we will move less cautiously if the cup is empty. Behind this decision making process, one needs to make predictions about 'what could go wrong' given the current observations, and decide what actions to take so that we do not end up with a bad outcome. Such capability is essential for humans and more generally, animals, to handle the ever-changing and unpredictable real-world.

Developing computational tools to equip robots with similar capability of perceiving and acting upon risks is a crucial step towards deploying robots in real-world applications, especially for those that require robots to work closely with people. However, this is a challenging problem for a few reasons. First, being able to detect that a situation has high risk requires the robot to know what would happen in the future given different plans and aggregate them to understand how many of them will end up being bad. Acquiring such predictive capability can be challenging for real robots. Second, the robot needs to intelligently select an action given the risk assessment such that it can make progress towards completing the task while ensuring safety of the robot.

Existing methods in this area can be broadly divided into two directions: model-free approach, and model-based approach. In model-free approach, the main focus has been on imposing constraints to the states or reward of the learning agent [98, 99]. Throughout the learning, the policy is updated to take the constraints into consideration and will satisfy the imposed constraints in expectation. However, during early stage of learning, the

policy may still violate the constraints, making it difficult to be applied to safety critical scenarios. In model-based approach, a model of the system dynamics is usually assumed to be available. Using these models, one can plan the controls using model-predictive control algorithms to incorporate safety constraints, or to derive theoretical safety bounds by leveraging techniques like control barrier functions. However, these methods are usually assumed to be lipschitz continuous to facilitate the analysis and may not be easy to obtain for general robotic systems.

As mentioned in the previous Chapters, physics-based simulation techniques provide a safe and efficient environment for robots to explore different scenarios it might encounter in the real-world, and allow them to experience potentially unsafe states that are not feasible in the real-world. In the previous Chapter, we introduced a series of algorithms for learning locomotion policies in simulation and transfer them to the real hardware. An interesting question then is, can we also leverage computer simulation to help robots learn the concepts of safety and if this knowledge of safety can be transferred to the real world? In this Chapter, we develop an algorithm that takes a step towards answering this question. We define the safety of a robot state as: the expected duration of time that the robot can stay within a manually designed safety region given a controller optimized to be safe. Our proposed algorithm consists of four major components. First, we train a task policy π_{task} in the source environments that aims to achieve optimal performance for the task. Based on states visited by the task policy, we train a second control policy π_{safe} in source environments that keeps the simulated robot from entering unsafe regions such as states where the robot falls on the ground. This safe policy is trained to work under different initial poses and dynamics parameters, therefore we name it universal safe policy (USP). After training a USP, we learn a predictive model that estimates how 'safe' a given observation is. One possible way to define this safety estimation model is to use the value function from the trained USP. However, the value function of USP does not take into account the actions from the task policy: a state that is safe to the USP may not be safe if we take an action

from the task policy. As such, we train a one-step safety critic model (OSSC) that predicts the 'safety' of an observation assuming we follow the task policy for one step. Finally, we devise a process to combine the task policy, the USP, and the OSSC to achieve safe transfer in a novel target environment.

We demonstrate our method in two tasks: transferring locomotion policies of Hopper and Walker2d from one simulator (Dart) to another (MuJoCo) and compared to a few baseline methods. We demonstrate that our method exhibit notable robustness to novel environments compared to the baseline methods.

6.2 Training Universal Safe Policy

In this work, we train a universal safe policy (USP) that is dedicated to maintain the robot within a designated safety region without considering any downstream tasks. Training of the USP can be performed using standard deep reinforcement learning algorithms with a reward encouraging the robot to stay in the safe region. However, one aspect that USP deviates from the typical motor skill learning scenario is that when we learn a policy for a particular task, we only need to concern the states that are relevant to the task, while a USP needs to work for all the states within the safety region. For high dimensional continuous control problems this will become infeasible.

Since our final goal is being able to transfer a task policy from a source environment to a target environment, we can relax the requirement to be that the USP should work well for states that are relevant to the task. However, we do not know what states the robot is going to visit during the transfer. In our work, we approximate this state distribution by leveraging the trained task policy in the source environment. Specifically, we run the task policy with added noise in the source environment to generate a number of states. This gives us a set of states that are directly related to the task in the source environment. We then add additional noise to the collected states to expand them to the nearby states for robustness. During the training of USP, we will randomly sample initial states from these

states and train the policy to maintain safety of the robot. Algorithm 8 provides a more detailed description of our training process for USP.

Algorithm 8 Learning USP

// USP is learned entirely in source environment
Randomly initialize the weights for the USP π_{safe}
Run π_{task} for L rollouts and collect the robot states S
For each state $\mathbf{s}$ in S, randomly perturb $\mathbf{s}$ and add it to S
for $i = 1 : K$ **do**
    Initialize rollout buffer R
    Randomly choose initial state $\mathbf{s}_0$ from S
    $t \leftarrow 0$
    while $t \leq MaxStep$ **do**
        $\mathbf{u}_t = \pi_{safe}(\mathbf{s}_t, \boldsymbol{\mu})$
        $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$
        $r, terminated = Reward(\mathbf{s}_t, \mathbf{a}_t)$
        Push $(\mathbf{s}_t, \mathbf{a}_t, r)$ into R
        if $terminated$ **then**
            Randomly choose initial state $\mathbf{s}_{t+1}$ from S
    Update π_{safe} with data in R using DRL algorithms
return π_{safe}

6.3 Training One-Step Safety Critic

After training the task policy and the USP, we want to combine them in order to achieve successful transfer to novel situations. One possible way is to take a weighted average of the actions from both policies to produce the final control signal to the robot. However, this is unlikely to work as blending the actions from two policies usually does not lead to an interpolation of the policies' behaviors. In our approach, we choose at each timestep one of the two policies to generate the actions for the robot. As a result, the robot will be switching between a "task mode" where actions are taken to complete the task and a "safe mode" where the robot is trying to stay in the safe region. The question then is, how do we determine the mode that the robot should be in? To select the mode for the robot, one can train a higher level policy that outputs the index of the policy to query. However, we find that this strategy is not effective in achieving successful transfer: if we train the high-

level policy in the target environment, it requires notable amount of data from the target environment without guarantees for safety, while if we learn the policy in the simulation, it will converge to using exclusively the task policy.

In this thesis, we propose to perform the policy selection by estimating how 'safe' a given state is: for a state that is safe, we will use the task policy to generate the action, while for an unsafe state we will use the USP policy. Doing this requires an estimation of the safety of a given state. A natural way to do this is to use the value function from the USP as it measures the long-term reward of the safety policy from a certain state. However, using the value function alone is not sufficient for ensuring safety during transfer. In particular, the value function of USP computes the accumulated reward assuming the robot is following actions from USP, while we also need to take actions from the task policy in order to achieve the task. To take the actions from the task policy into consideration, we learn a second value function that computes the safety estimation for a state if we take one step from the task policy and then follow the USP. This will give us a better estimation of the safety at a certain robot state. We call this new model one-step safety critic (OSCC). In our implementation, we first re-train the value function of the USP to predict the normalized time to failure: $\text{nttf}(\mathbf{o}, \pi_{safe}) = \frac{\text{ttf}(\mathbf{o}, \pi_{safe})}{H}$, where $\text{ttf}(\cdot)$ is the time-to-failure from the observation $\mathbf{o}$ using the USP π_{safe} and H is the maximum horizon of a rollout. This gives us a more interpretable model output. A more detailed description of the algorithm can be found in Algorithm 9.

6.4 Safety-aware Policy Transfer

In the ideal case where OSSC can precisely predict the time-to-failure for an observation, one can manually choose a threshold κ for switching between using π_{task} and π_{safe}. For example, one may choose a κ close to one, meaning that we will use the action from the task policy only if the robot is predicted to be safe for the entire rollout horizon. In practice, however, OSSC will not be exact due to the training error and the noise in the observations.

Algorithm 9 Learning OSSC

// OSSC is trained using source environment only
Randomly initialize the weights for the OSSC ψ
Retrieve or re-train the value function $V : \mathbf{o} \mapsto v$ from USP
Run USP for M rollouts with initial state sampled from S (Algorithm 8)
Store the generated tuples $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})$ to a buffer B
for each $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})$ in B **do**
    Add $\mathbf{o}_t$ to the training input set T_{input}
    Add $V(\mathbf{o}_{t+1})$ to the training label set T_{label}
Optimize ψ using stochastic gradient descent with T_{input} and T_{label}
return ψ

Furthermore, because the OSSC model is trained entirely in the source environments, we should not assume its accuracy in the target environment. As a result, we would need to search for a threshold κ that can achieve the best transfer performance in the target environment. In addition, we found that using a single threshold κ to perform the policy selection can sometimes lead to oscillations between the two policies, leading to states not familiar to either of them. As such, we define two thresholds κ_{task} and κ_{safe} for the two policies respectively. When the robot is running actions from π_{task}, it will make a switch to the USP policy if the safety estimation by OSSC is lower than κ_{task}. Similarly, the robot will switch to the task policy π_{task} from USP π_{safe} if the estimated safety value is higher than κ_{safe}. When put together, we obtain a stateful control policy whose actions are determined as follows:

$$\pi_{combine}(\mathbf{o}, mode) = \begin{cases} \pi_{safe}(\mathbf{o}), & \text{if } \psi(\mathbf{o}) < \kappa_{safe} \text{ and } mode = SAFE \\ \pi_{task}(\mathbf{o}), & \text{if } \psi(\mathbf{o}) > \kappa_{task} \text{ and } mode = TASK, \end{cases}$$

$$\text{where } mode = \begin{cases} SAFE, & \text{if } \psi(\mathbf{o}) < \kappa_{task} \text{ and } mode = TASK \\ TASK, & \text{if } \psi(\mathbf{o}) > \kappa_{safe} \text{ and } mode = SAFE, \end{cases}.$$

Searching for the two thresholds κ_{safe} and κ_{task} can be done using strategy optimization as in our previous work 5.3. However, doing this requires deploying the policies in the target environment, without any guarantees about the safety of the resulting policy, which

is in conflict with the purpose of training the safety policy. In this thesis, we design a safety-aware policy transfer approach that minimizes the possible failure cases when transferring the trained models to a new environment. Our core idea is that we start with the most conservative strategy (using π_{safe} only) and gradually relax the thresholds to allow the task policy to take actions when the robot is predicted to be in a safe state. Specifically, we first fix $\kappa_{safe} = 1$ and search for the smallest κ_{task} that can keep the robot in the safe region. This step approximately finds the boundary where the task policy cannot transfer to the target environment and needs assistance from the USP. Using this threshold alone does not allow the robot to switch from USP to task policy, which is overly conservative. Therefore, we perform a second phase of search where we fix the optimized κ_{task} and gradually reduce κ_{safe} until the resulting policy leads the robot into an unsafe state. After the search, we will return the κ_{task} and κ_{safe} that achieve the best task performance in the target environment. Furthermore, we set a reward threshold $\bar{R}$ that terminates the search process if the current policy reaches satisfying performance. As a result, the robot will enter unsafe region at most twice during the transfer. Algorithm 10 describes our safety-aware transfer algorithm.

6.5 Results

To evaluate our algorithm, we take a similar strategy as in Chapter 5, where we validate the algorithm by designing a set of sim-to-sim transfer tasks using two different simulators, Dart [5] and MuJoCo [6]. Appendix A provides a comparison between the two simulators and discusses the main challenges in tranferring from one to the other. We aim to answer the following questions in our experiments: 1) Does our method achieve safe transfer to novel environments while making progress? 2) Given the same amount of allowance for failure in target environment, does our method outperform alternative methods? and 3) Does our transfer scheme produce reasonable scheduling between the task and safe policy?

To answer these questions, we build our experiment environments based on two of the tasks from Chapter 5: training the Hopper and Walker2d to run forward in Dart and

Algorithm 10 Safety-aware Policy Transfer

Input: π_{task}, π_{safe}, ψ, reward threshold $\bar{R}$, search interval Δ, minimum threshold κ_{min}
Initialize $\kappa_{task} = 1.0$ and $\kappa_{safe} = 1.0$
while $\kappa_{task} > \kappa_{min}$ **do**
    Evaluate performance R of $\pi_{combine}$ with κ_{task} and κ_{safe}
    if $R > \bar{R}$ **then**
        Break
    if Robot unsafe **then**
        $\kappa_{task} = \kappa_{task} + \Delta$
        Break
    $\kappa_{task} = \kappa_{task} - \Delta$
while $\kappa_{safe} > \kappa_{task}$ **do**
    Evaluate performance R of $\pi_{combine}$ with κ_{task} and κ_{safe}
    if $R > \bar{R}$ **then**
        Break
    if Robot unsafe **then**
        $\kappa_{safe} = \kappa_{safe} + \Delta$
        Break
    $\kappa_{safe} = \kappa_{safe} - \Delta$
return κ_{task}, κ_{safe}

transfer to a different environment (Figure 6.1). Details for these two environments can be found in Appendix B. For the Hopper environment, we randomize 10 dynamics parameters during training: the mass of each segment of the robot, the damping coefficient of each joint, and the friction and restitution coefficient between the foot and the ground. For the Walker2d environment, we randomize the same set of dynamics parameters as in the Hopper environment. In addition, we also add random perturbations to both robots during training. We found that doing this can further improve the robustness of training in all methods. For defining the safety region for training universal safe policies (USP), we use the termination criteria from the environments, which is also used in training the task policies.

We use Proximal Policy Optimization (PPO) for training task policies and USPs. Each policy is trained with $15,000,000$ samples with $30,000$ samples collected in each training iteration.

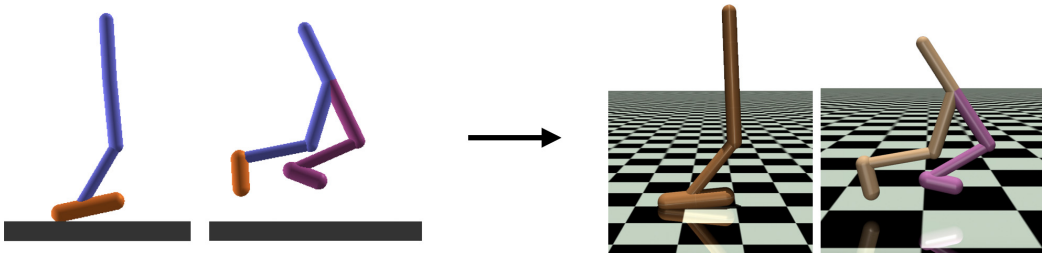


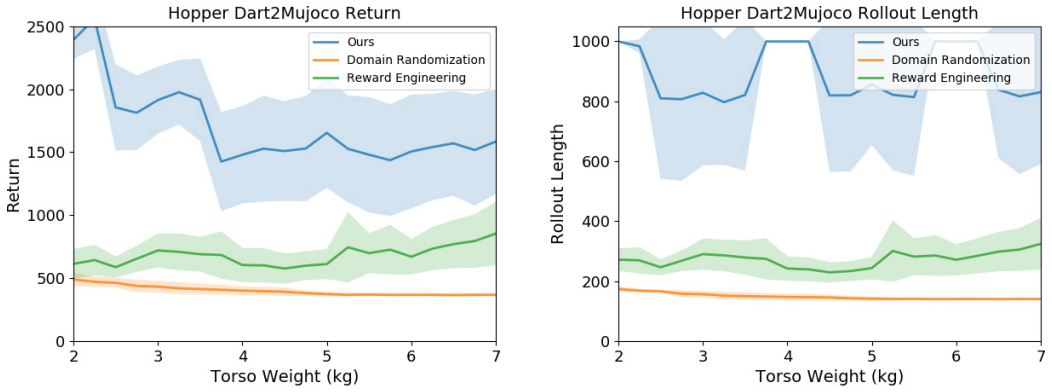
Figure 6.1: The two environments used in our experiments. We transfer locomotion policies from Dart (left) to MuJoCo (right). The training and testing environments are different in their contact modeling, joint limit modeling, armature modeling, and latency modeling.

6.5.1 Does our method achieve safe transfer to novel environments?

To evaluate if our method can indeed achieve safety transfer, we adapt the trained locomotion policies to target environments that are significantly different from the training environments. In addition to the differences between the simulators we use as discussed in Appendix A, we also add latency to the target environment (no latency is used during training). We test our method in the target environment for a set of dynamics parameters to estimate the overall performance of our method.

For the Hopper environment in MuJoCo, we evaluate the policy performance by varying the weight of the torso link between $[2.0, 7.0]$ kg. We also add a latency of 8 ms in the testing environment, which was not used in our previous experiments (5.3). The results can be found in Figure 6.2. We can see that our method can successfully find a policy that can safely control the robot to walk in the testing environment. Note that for some testing environment variations as shown in Figure 6.2 (b) the resulting policy does not reach the end of the rollout. This is because algorithm outputs the policy obtaining highest return during the adaptation process, as a result, a robot that hops forward but falls near the end of the rollout is considered better than a robot standing still. We expect this will be mitigated if we use a longer time horizon for testing. For the return of the policy (Figure 6.2 (a)), our method is able to achieve reasonable task performance (an oracle model trained with $1,000,000$ samples achieves a return of around 2000).

For the Walker2d environment in MuJoCo, we vary two dynamics parameters during



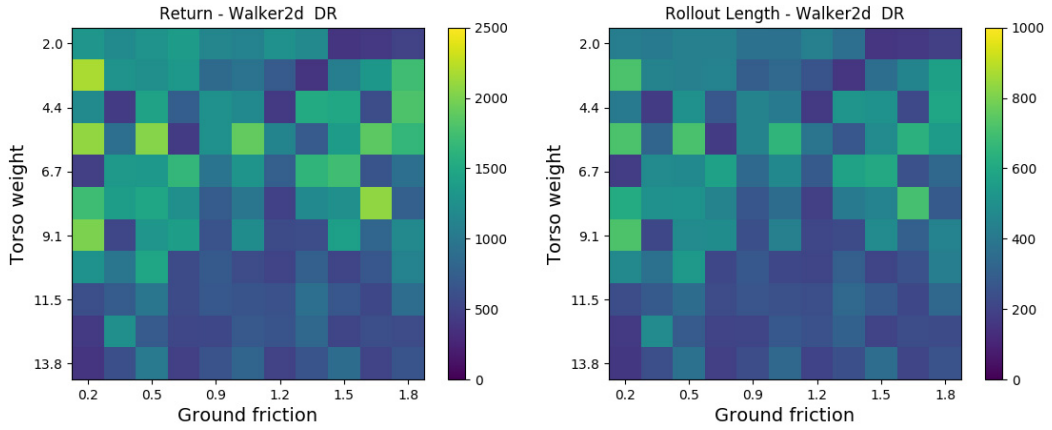
(a) (b)

Figure 6.2: Transfer performance for the Hopper example. (a) comparison of total returns of the policies. (b) comparison of the rollout length of the policies. Shaded area denotes one standard deviation.

testing: weight of the foot link in $[2.0, 15.0]$ kg and ground friction coefficient in $[0.5, 2.0]$. We add a larger latency of 16 ms during testing (previously 8 ms). Figure 6.4 shows the performance of our approach for the Walker2d environment. We see that our method again can transfer the policy to the testing environment with good safety measurement. Furthermore, for most of the variations in the testing environments, we are able to achieve reasonable task performances. For some variations that achieve a return near or below 1000 (e.g. the point at torso weight 11.5 kg and ground friction 0.9), it indicates that our method is not able to find a policy that achieves both good task performance and safety. Consequently, our algorithm produces a policy that keeps the robot safe, but does not make much progress for the locomotion task.

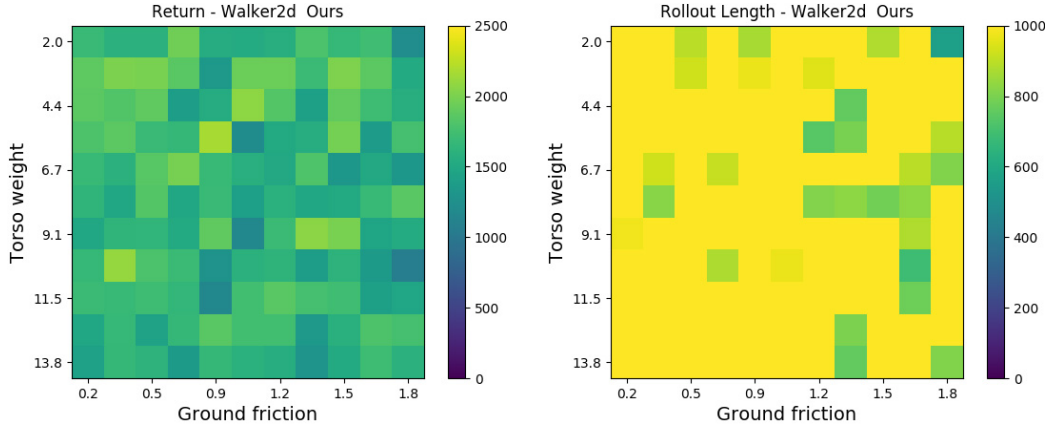
6.5.2 How does our method compare to alternative methods?

We compare our method to training a domain randomization (DR) policy alone. A DR policy is trained to output actions that can work for all training variations without distinguishing them, thus a DR policy is robust to different training environments. However, when we apply a DR policy to notably different environments, as shown in Figure 6.2 (b) and Figure 6.3, it is not able to successfully control the robot to complete the task and in most scenarios the robot will enter the unsafe region such as falling to the ground. Due



(a) (b)

Figure 6.3: Result for the Walker2d example using Domain Randomization (DR).



(a) (b)

Figure 6.4: Result for the Walker2d example using our proposed algorithm.

to that the robots usually falls to the ground quickly, the total return we obtain from DR policies are in general worse than the policies from our approach.

For the Hopper example, we also compare our method to training a DR policy with reward engineering: we increased the alive bonus in the reward function from 1 to 4 during training of the policy. As shown in Figure 6.2, doing reward engineering can indeed improve the performance of the DR policy. However, the performance is notably worse than our approach and finding a good reward function for achieving both safety and task performance can be challenging.

6.5.3 Does our transfer scheme produce reasonable scheduling between the task and safe policy?

Finally, we examine the policy scheduling strategy emerged from our method. We take the MujoCo Hopper environment with torso weight of 2.5 kg and deployed our optimized model with adapted safety thresholds. To investigate what kind of strategy the model has learned to select between task and safety policies, we collect the observations and the policy selection throughout one trajectory and train a logistic regression model to predict the policy selection from the observations. From the trained coefficients of the logistic regression model, we find that torso pitch and forward velocity of the Hopper are the two most predictive features for the policy selection. To further examine the results, we plot the scheduling of the policy and the two features in Figure 6.5. For better visualization of the task scheduling, we scale down the values for torso pitch and forward velocity. We can see from the plot that when the forward velocity of the robot is high and is tilting forward (positive torso pitch), our method would choose to use the safety policy. This makes sense because the combination of high forward velocity and forward tilting indicates that the robot might fall forward.

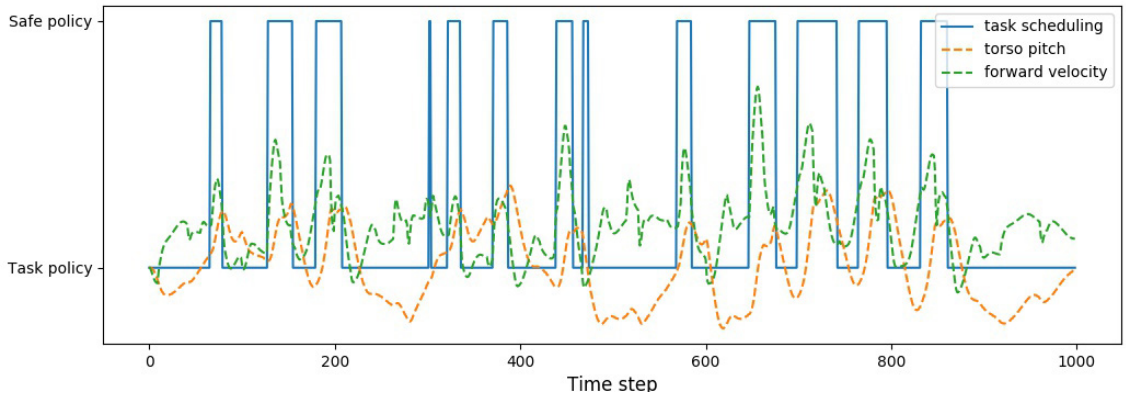


Figure 6.5: Policy scheduling over one trajectory for the Hopper environment. Dashed lines are the two most predictive features and are scaled down for better visualization.

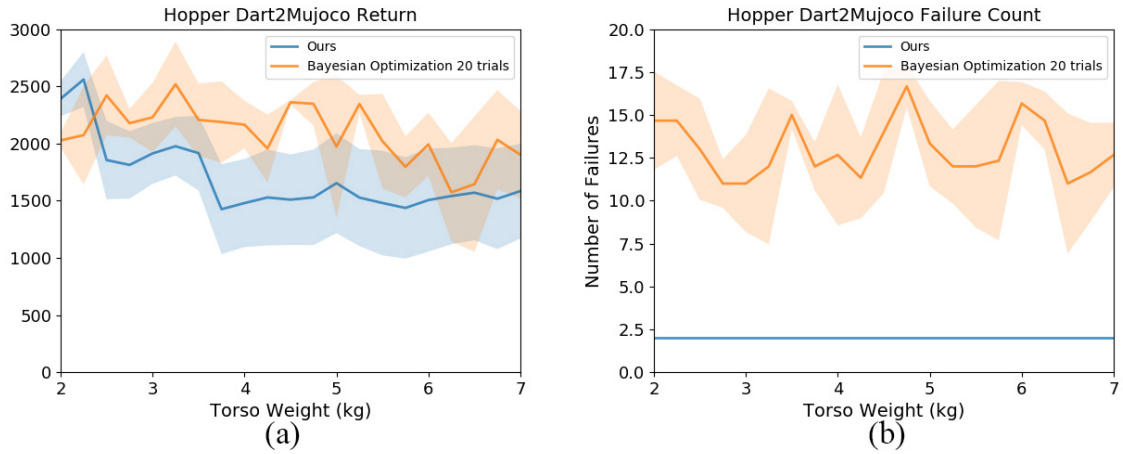


Figure 6.6: Comparison of our method with Bayesian Optimization for the Hopper example.

6.6 Discussion

In this Chapter we have presented a transfer learning algorithm that can safely transfers a simulation-trained control policy to a novel target environment. Our algorithm trains three models in the source environment: a task policy that completes the desired task, a universal safe policy that keeps the robot within the safety region, and a one-step safety critic that estimates the time-to-failure from the current robot observation. We then develop an algorithm for transferring these three models to the target environment. Our algorithm, by design, ensures that the robot can enter the unsafe region no more than twice. We demonstrate that our proposed algorithm can overcome large modeling errors that resembles the ones seen in sim-to-real transfer scenarios.

Our method does have a few limitations. First, although our method can robustly transfer policies to new environments, we usually obtain results that are sub-optimal in the task performance. For example, we found that using Bayesian Optimization to search for the thresholds κ_{safe} and κ_{task} can lead to better performance as shown in Figure 6.6 at the cost of higher number of failure trials during transfer. Furthermore, the resulting policy can sometimes be over-conservative, leading to robots moving very slowly or standing still. One possible direction to achieve both safe and high-performance transfer would be to

combine our method with our previous strategy optimization-based method such that we search for both the latent input $\boldsymbol{\mu}$ to the policy and the thresholds κ_{task} and κ_{safe}. The challenge would be that this would require a large amount of samples in the target environment. Second, our method assumes that the safety of the robot can be estimated from the current observation from the robot. This assumption can be invalidated by latency in the system or hidden states of the robot that are not available in the observations. Incorporating memory in the safe policy may mitigate this issue. Furthermore, our current algorithm generates up to two failed trials in the target environment, which is significantly lower than methods that do not take safety into consideration. However, for safety-critic applications such as assitive robots, we cannot afford any failure for the robot. One possible way to address this issue is to combine our method with model-based safe DRL methods, for which a safety metric can be bounded analytically.

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this thesis, we have presented a set of algorithms that make progress towards automated and generalizable locomotion learning for simulated characters and real robots. Our algorithms achieve automated learning of locomotion skills using Deep Reinforcement Learning algorithms. By developing novel learning-based algorithms that can handle difficult optimization problems and a loss term to regularize the resulting motion, we obtain plausible locomotion gaits without relying on specific prior knowledge such as finite state machines (FSMs) or motion data. In addition to automated locomotion learning, we also make contributions in achieving generalizable locomotion learning. We propose a set of transfer learning algorithms that learn a family of control policies in the training environment and intelligently select the best policy to use in the testing environment. Using our algorithm, we achieve successful transfer from simulation to real legged robots with a handful of trials ($15 - 25$). Moreover, we investigate the problem of safety when transferring a policy to a new environment. By learning a controller dedicated to maintain the safety of the robot and developing a transfer algorithm for carefully activating the safety controller, we demonstrate successful transfer of locomotion policies to significantly different environments with at most two failure trials. These algorithms show the possibility of using a learning-based framework to enable robots and simulated character to automatically acquire complex skills and generalize them to novel situations.

In Chapter 4, we introduce a reinforcement learning approach for creating low-energy, symmetric, and speed-appropriate locomotion gaits. One element of this approach is to provide virtual assistance to help the character learn to balance and to reach a target speed.

The second element is to encourage symmetric behavior through the use of an additional loss term. When used together, these two techniques provide a method of automatically creating locomotion controllers for arbitrary character body plans. We tested our method on the lower half of a biped, a full humanoid, a quadruped, an a hexapod and demonstrated learning of locomotion gaits that resemble the natural motions of humans and animals. Because our method generalizes to other body plans, an animator can create locomotion controllers for characters and creatures for which there is no existing motion data.

In Chapter 5, we present a general transfer learning framework, where a family of control policies is trained to capture different variations in the training environment and the best one is selected to be used in the target environment. We demonstrate that by representing the family of control policies as a universal policy (UP) and training an online system identification model (OSI), we obtain a policy that can adapt to changes in the environments such as different ground frictions. Though OSI can handle changes in the dynamics, its generalization ability is limited within the training dynamics. To allow the trained policy to generalize to environments outside the training domain, we propose using strategy optimization (SO) that directly optimizes the policy input directly in the testing environment. We demonstrate that this significantly improves the generalization capability of the trained control policies. We apply the idea of SO to train locomotion controllers for a biped robot, Robotis Darwin OP2, and a quadruped robot, Ghost Robotics Minitaur and transfer them to the real robot hardwares. When training the locomotion policy for Darwin OP2, we introduce a projection network that projects the dynamics parameters into a lower dimensional latent space before it is fed into the control policy. This allows us to perform SO in a lower dimensional space, notably reducing the number of trials needed for sim-to-real transfer. For training the locomotion policy for Minitaur, we further developed meta strategy optimization (MSO) that uses SO to obtain the latent space during both training and testing. This allows us to train a latent policy space that is more suitable for adaptation. We show that MSO can achieve successful sim-to-real transfer of a locomotion policy for

Minitaur. In addition, MSO can also transfer the trained locomotion policy to new tasks such as walking on a slope, while the idea based on projection network were not able to achieve successful transfer.

In Chapter 6, we develop a safe transfer learning algorithm for improving safety when transferring locomotion policies to notably different environments not seen during training. Our key observation is that a policy dedicated to maintain the robot safe (e.g. a balancing policy that keeps robot from falling) can usually generalize better to novel situations than a policy optimizing the task reward (e.g. walking forward). Based on this observation, we propose to train a universal safe policy (USP) that keeps the robot within the safety region as well as a one-step safety critic that estimates the safety of a given robot state when applying the trained USP. When transferring these source environment-trained models to the target environment, we introduce a transfer scheme that finds the best combination of task policy and safe policy in the target environment. Our transfer scheme finds a safe control policy with at most two failure trials in the target environment.

7.2 Future Work

This dissertation presents a set of computational tools for teaching computer simulated characters and real robots to walk. Our research takes a step toward the goal of creating learning agents that can acquire complex motor skills in an automated and generalizable way and opens many exciting future research directions in both short term and long term. For example, although our locomotion learning algorithm presented in Chapter 4 demonstrate more realistic locomotion gaits comparing to existing work in DRL, the quality of the motion is still not on a par with previous work in computer animation that exploits real-world data. By incorporating biological-based modeling, Jiang *et al.* [119] achieved improved motion quality with reduced requirements for reward engineering. Nevertheless, there is still room for improvement. Understanding and bridging the gap of the perceived naturalness between our approach and real animal and human motions is thus an interest-

ing and important direction to pursue. Another promising future direction is to improve our existing transfer learning algorithms. Our current algorithms in Chapter 5 can adapt to new environments with high performance, but requires multiple failed trials to learn the performance landscape, while our algorithm in Chapter 6 achieves safe transfer to novel environments at the cost of task performance. Therefore, an interesting direction to investigate is to achieve both safe and optimal performing transfer to real robots by combining our methods in Chapter 5 and Chapter 6.

In the long term, we envision a few important research directions to pursue while projecting the possible future technological advancements. First, we demonstrate in our work (Chapter 4) that by imposing general prior knowledge about locomotion skills such as symmetry and low-energy for locomotion, we can get a locomotion learning algorithm that obtains good optimization performance while being general to different characters. However, to come up with these prior knowledge and to incorporate them into a motor learning framework still relies heavily on human researchers, making it challenging to be applied to a broader set of tasks, especially those with more abstract goals such as "cleaning the room". We envision that an interesting future direction is to develop algorithms that can automatically extract and incorporate general prior knowledge for motor tasks from unstructured human data such as images, videos, and texts. This will likely benefit from future developments in computer vision and natural language processing.

Second, the most impressive skills seen on existing robots usually rely heavily on offline computing: the controller or the policy is optimized on a separate platform and then deployed on the robot. With advancements in hardware, we expect to see nimble robots equipped with powerful computing capabilities that allow them to incorporate new experience and perform complex computations internally while performing the motor skills. I believe this is an important step towards creating robots that can achieve life-long learning - a key ability for both autonomy and generalization. Investigating learning algorithms that enables such capability is thus an exciting direction to pursue.

Appendices

APPENDIX A

COMPARISON BETWEEN DART AND MUJOCO SIMULATOR

DART [5] and MuJoCo [6] are both physically-based simulators that computes how the state of virtual character or robot evolves over time and interacts with other objects in a physical way. Both of them have been demonstrated for transferring controllers learned for a simulated robot to a real hardware [58, 134], and there has been work trying to transfer policies between DART and MuJoCo [135]. The two simulators are similar in many aspects, for example both of them uses generalized coordinates for representing the state of a robot. Despite the many similarities between DART and MuJoCo, there are a few important differences between them that makes transferring a policy trained in one simulator to the other challenging. For the examples of DART-to-MuJoCo transfer presented in this dissertation, there are three major differences as described below:

1. Contact Handling

   Contact modeling is important for robotic control applications, especially for locomotion tasks, where robots heavily rely on manipulating contacts between end-effector and the ground to move forward. In DART, contacts are handled by solving a linear complementarity problem (LCP) [136], which ensures that in the next timestep, the objects will not penetrate with each other, while satisfying the laws of physics. In MuJoCo, the contact dynamics is modeled using a complementarity-free formulation, which means the objects might penetrate with each other. The resulting impulse will increase with the penetration depth and separate the penetrating objects eventually.

2. Joint Limits

   Similar to the contact solver, DART tries to solve the joint limit constraints exactly

so that the joint limit is not violated in the next timestep, while MuJoCo uses a soft constraint formulation, which means the character may violate the joint limit constraint.

3. Armature

   In MuJoCo, a diagonal matrix $\sigma\mathbb{I}_n$ is added to the joint space inertia matrix that can help stabilize the simulation, where $\sigma \in \mathbb{R}$ is a scalar named Armature in MuJoCo and $\mathbb{I}_n$ is the $n \times n$ identity matrix. This is not modeled in DART.

To illustrate how much difference these simulator characteristics can lead to, we compare the Hopper example in DART and MuJoCo by simulating both using the same sequence of randomly generated actions from an identical state. We plot the linear position and velocity of the torso and foot of the robot, which is shown in Figure A.1. We can see that due to the differences in the dynamics, the two simulators would control the robot to reach notably different states even though the initial state and control signals are identical.

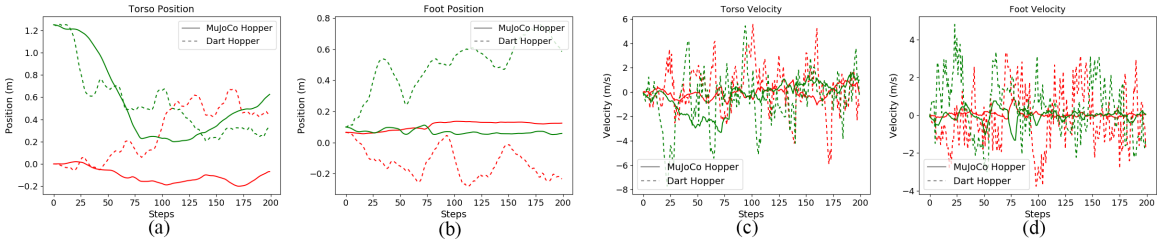


Figure A.1: Comparison of DART and MuJoCo environments under the same control signals. The red curves represent position or velocity in the forward direction and the green curves represent position or velocity in the upward direction.

APPENDIX B

DETAILS FOR SIM-TO-SIM TRANSFER ENVIRONMENTS

B.1 Environment Details

The observation space, action space and the reward function used in all of our examples can be found in Table **??**. For the Walker2d environment, we found that with the original environment settings in OpenAI Gym, the robot sometimes learn to hop forward, possibly due to the ankle being too strong. Therefore, we reduce the torque limit of the ankle joint in both DART and MuJoCo environment for the Walker2d problem from $[-100, 100]$ to $[-20, 20]$. We found that with this modification, we can reliably learn locomotion gaits that are closer to a human running gait.

Below we list the dynamic randomization settings used in our experiments. Table B.1, Table B.2 and Table B.3 shows the range of the randomization for different dynamic parameters in different environments. For the quadruped example, we used the same settings as in Tan *et al.* [58].

Table B.1: Dynamic Randomization details for Hopper

Dynamic Parmeter	Range
Friction Coefficient	$[0.2, 1.0]$
Restitution Coefficient	$[0.0, 0.3]$
Mass	$[2.0, 15.0]$kg
Joint Damping	$[0.5, 3]$
Joint Torque Scale	$[50\%, 150\%]$

B.2 Simulated Reality Gaps

To evaluate the ability of our method to overcome the modeling error, we designed six types of modeling errors. Each example shown in our experiments contains one or more

Table B.2: Dynamic Randomization details for Walker2d

Dynamic Parmeter	Range
Friction Coefficient	$[0.2, 1.0]$
Restitution Coefficient	$[0.0, 0.8]$
Joint Damping	$[0.1, 3.0]$

Table B.3: Dynamic Randomization details for HalfCheetah

Dynamic Parmeter	Range
Friction Coefficient	$[0.2, 1.0]$
Restitution Coefficient	$[0.0, 0.5]$
Mass	$[1.0, 15.0]$kg
Joint Torque Scale	$[30\%, 150\%]$

modeling errors listed below.

1. DART to MuJoCo

   For the Hopper, Walker2d and HalfCheetah example, we trained policies that transfers from DART environment to MuJoCo environment. As discussed in Appendix A, the major differences between DART and MuJoCo are contacts, joint limits and armature.

2. Latency

   The second type of modeling error we tested is latency in the signals. Specifically, we model the latency between when an observation o is sent out from the robot, and when the action corresponding to this observation $a = \pi(o)$ is executed on the robot. When a policy is trained without any delay, it is usually very challenging to transfer it to problems with delay added. The value of delay is usually below 50ms and we use 8ms and 50ms in our examples.

3. Actuator Modeling Error

As noted by Tan *et al.* [58], error in actuator modeling is an important factor that contributes to the reality gap. They solved it by identifying a more accurate actuator model by fitting a piece-wise linear function for the torque-current relation. We use their identified actuator model as the ground-truth target environment in our experiments and used the ideal linear torque-current relation in the source environments.

4. Foot Mass

   In the example of Walker2d, we vary the mass of the right foot on the robot to create a family of target environments for testing. The range of the torso mass varies in $[2, 9]$kg.

5. Terrain Slope

   In the example of HalfCheetah, we vary the slope of the ground to create a family of target environments for testing. This is implemented as rotating the gravity direction by the same angle. The angle varies in the range $[-0.18, 0.0]$ radians.

6. Rigid to Deformable

   The last type of modeling error we test is that a deformable object in the target environment is modeled as a rigid object in the source environment. The deformable object is modeled using the soft shape object in DART. In our example, we created a deformable box of size $0.5m \times 0.19m \times 0.13m$ around the foot of the Hopper. We set the stiffness of the deformable object to be $10,000$ and the damping to be 1.0. We refer readers to Jain *et al.* [129] for more details of the softbody simulation.

REFERENCES

[1] H. L. Lee, Y. Chen, B. Gillai, and S. Rammohan, "Technological disruption and innovation in last-mile delivery," *Value Chain Innovation Initiative*, 2016.

[2] C. Loughlin, Z. Wang, and H. Gu, "A review of locomotion mechanisms of urban search and rescue robot," *Industrial Robot: An International Journal*, 2007.

[3] K. Hauser, T. Bretl, J.-C. Latombe, and B. Wilcox, "Motion planning for a six-legged lunar robot," in *Algorithmic Foundation of Robotics VII*, Springer, 2008, pp. 301–316.

[4] K. A. Witte, P. Fiers, A. L. Sheets-Singer, and S. H. Collins, "Improving the energy economy of human running with powered and unpowered ankle exoskeleton assistance," *Science Robotics*, vol. 5, no. 40, 2020.

[5] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "Dart: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[6] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, IEEE, 2012, pp. 5026–5033.

[7] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation in robotics, games and machine learning*. 2016-2017.

[8] R. M. Alexander, *Principles of animal locomotion*. Princeton University Press, 2003.

[9] G. Castillo and M. Neff, "What do we express without knowing?: Emotion in gesture," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 702–710.

[10] M. Neunert, T. Boaventura, and J. Buchli, "Why off-the-shelf physics simulators fail in evaluating feedback controller performance-a case study for quadrupedal robots," in *Advances in Cooperative Robotics*, World Scientific, 2017, pp. 464–472.

[11] W. Yu, G. Turk, and C. K. Liu, "Learning symmetric and low-energy locomotion," *ACM Transactions on Graphics (Proc. SIGGRAPH 2018)*, vol. 37, no. 4, 2018.

[12] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," in *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, 2017.

[13] W. Yu, C. K. Liu, and G. Turk, "Policy transfer with strategy optimization," in *International Conference on Learning Representations*, 2019.

[14] W. Yu, V. C. Kumar, G. Turk, and C. K. Liu, "Sim-to-real transfer for biped locomotion," *arXiv preprint arXiv:1903.01390*, 2019.

[15] W. Yu, J. Tan, Y. Bai, E. Coumans, and S. Ha, "Learning fast adaptation with meta strategy optimization," *arXiv preprint arXiv:1909.12995*, 2019.

[16] B. A. Silva, C. T. Gross, and J. Gräff, "The neural circuits of innate fear: Detection, integration, action, and memorization," *Learning & Memory*, vol. 23, no. 10, pp. 544–555, 2016.

[17] J. K. Hodgins, "Biped gait transitions," in *ICRA*, 1991, pp. 2092–2097.

[18] J. Hodgins and M. H. Raibert, "Biped gymnastics," *Dynamically Stable Legged Locomotion*, vol. 79, 1988.

[19] J. K. Hodgins and M. H. Raibert, "Adjusting step length for rough terrain locomotion," *Dynamically Stable Legged Locomotion*, p. 27, 1991.

[20] M. H. Raibert, *Legged robots that balance*. MIT press, 1986.

[21] R. R. Playter and M. H. Raibert, "Control of a biped somersault in 3d," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, vol. 1, 1992, pp. 582–589.

[22] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien, "Animating human athletics," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995, pp. 71–78.

[23] K. Yin, K. Loken, and M. van de Panne, "Simbicon: Simple biped locomotion control," *ACM Trans. Graph.*, vol. 26, no. 3, Jul. 2007.

[24] J.-c. Wu and Z. Popović, "Realistic modeling of bird flight animations," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 3, pp. 888–895, 2003.

[25] J. Tan, Y. Gu, G. Turk, and C. K. Liu, "Articulated swimming creatures," in *ACM SIGGRAPH 2011 papers*, ser. SIGGRAPH '11, Vancouver, British Columbia, Canada: ACM, 2011, 58:1–58:12, ISBN: 978-1-4503-0943-1.

[26] J. Tan, Y. Gu, C. K. Liu, and G. Turk, "Learning bicycle stunts," *ACM Trans. Graph.*, vol. 33, no. 4, 50:1–50:12, 2014.

[27] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien, "Animating human athletics," in *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '95, New York, NY, USA: ACM, 1995, pp. 71–78, ISBN: 0-89791-701-4.

[28] S. Jain, Y. Ye, and C. K. Liu, "Optimization-based interactive motion synthesis," *ACM Transaction on Graphics*, vol. 28, no. 1, pp. 1–10, 2009.

[29] M. de Lasa, I. Mordatch, and A. Hertzmann, "Feature-based locomotion controllers," *ACM Trans. Graph.*, vol. 29, no. 4, 131:1–131:10, Jul. 2010.

[30] T. Geijtenbeek, M. van de Panne, and A. F. van der Stappen, "Flexible muscle-based locomotion for bipedal creatures," *ACM Trans. Graph.*, vol. 32, no. 6, 206:1–206:11, Nov. 2013.

[31] J. M. Wang, S. R. Hamner, S. L. Delp, and V. Koltun, "Optimizing locomotion controllers using biologically-based actuators and objectives," *ACM Trans. Graph.*, vol. 31, no. 4, 25:1–25:11, Jul. 2012.

[32] S. Coros, P. Beaudoin, and M. van de Panne, "Generalized biped walking control," in *ACM SIGGRAPH 2010 Papers*, ser. SIGGRAPH '10, Los Angeles, California: ACM, 2010, 130:1–130:9, ISBN: 978-1-4503-0210-4.

[33] S. Coros, A. Karpathy, B. Jones, L. Reveret, and M. van de Panne, "Locomotion skills for simulated quadrupeds," *ACM Transactions on Graphics*, vol. 30, no. 4, Article TBD, 2011.

[34] M. L. Felis and K. Mombaur, "Synthesis of full-body 3-d human gait using optimal control methods," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, IEEE, 2016, pp. 1560–1566.

[35] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Optimizing walking controllers," *ACM Trans. Graph.*, vol. 28, no. 5, 168:1–168:8, Dec. 2009.

[36] W.-L. Ma, A. Hereid, C. M. Hubicki, and A. D. Ames, "Efficient hzd gait generation for three-dimensional underactuated humanoid running," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 5819–5825.

[37] A. Hereid, S. Kolathaya, and A. D. Ames, "Online hybrid zero dynamics optimal gait generation using legendre pseudospectral optimization," in *IEEE Conference on Decision and Control*, 2016.

[38] Y. Gong, R. Hartley, X. Da, A. Hereid, O. Harib, J.-K. Huang, and J. Grizzle, "Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway," in *2019 American Control Conference (ACC)*, IEEE, 2019, pp. 4559–4566.

[39] K. Wampler, Z. Popović, and J. Popović, "Generalizing locomotion style to new animals with inverse optimal regression," *ACM Trans. Graph.*, vol. 33, no. 4, 49:1–49:11, Jul. 2014.

[40] Y. Lee, M. S. Park, T. Kwon, and J. Lee, "Locomotion control for many-muscle humanoids," *ACM Trans. Graph.*, vol. 33, no. 6, 218:1–218:11, Nov. 2014.

[41] M. da Silva, Y. Abe, and J. Popović, "Interactive simulation of stylized human locomotion," *ACM Trans. Graph.*, vol. 27, no. 3, 82:1–82:10, Aug. 2008.

[42] Y. Ye and C. K. Liu, "Optimal feedback control for character animation using an abstract model," *ACM Trans. Graph.*, vol. 29, no. 4, 74:1–74:9, Jul. 2010.

[43] U. Muico, Y. Lee, J. Popović, and Z. Popović, "Contact-aware nonlinear control of dynamic characters," *ACM Trans. Graph.*, vol. 28, no. 3, 81:1–81:9, Jul. 2009.

[44] K. W. Sok, M. Kim, and J. Lee, "Simulating biped behaviors from human motion data," in *ACM SIGGRAPH 2007 Papers*, ser. SIGGRAPH '07, San Diego, California: ACM, 2007.

[45] C. K. Liu, A. Hertzmann, and Z. Popović, "Learning physics-based motion style with nonlinear inverse optimization," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1071–1081, Jul. 2005.

[46] Y. Lee, S. Kim, and J. Lee, "Data-driven biped control," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, p. 129, 2010.

[47] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv preprint arXiv:2004.00784*, 2020.

[48] M. Al Borno, M. De Lasa, and A. Hertzmann, "Trajectory optimization for full-body movements with complex contacts," *IEEE transactions on visualization and computer graphics*, vol. 19, no. 8, pp. 1405–1414, 2013.

[49] K. Wampler and Z. Popović, "Optimal gait and form for animal locomotion," in *ACM SIGGRAPH 2009 Papers*, ser. SIGGRAPH '09, New Orleans, Louisiana: ACM, 2009, 60:1–60:8, ISBN: 978-1-60558-726-4.

[50] I. Mordatch, J. M. Wang, E. Todorov, and V. Koltun, "Animating human lower limbs using contact-invariant optimization," *ACM Trans. Graph.*, vol. 32, no. 6, 203:1–203:8, Nov. 2013.

[51] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Trans. Graph.*, vol. 31, no. 4, 43:1–43:8, Jul. 2012.

[52] S. Ha and C. K. Liu, "Iterative training of dynamic skills inspired by human coaching techniques," *ACM Transactions on Graphics*, vol. 34, no. 1, 2014.

[53] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[54] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[55] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[56] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[57] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[58] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, 2018.

[59] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, eaau5872, 2019.

[60] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, ICRA, 2018, pp. 1–8.

[61] OpenAI, : M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning Dexterous In-Hand Manipulation," *ArXiv e-prints*, Aug. 2018. arXiv: 1808.00177.

[62] J. P. Hanna and P. Stone, "Grounded action transformation for robot learning in simulation," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, S. P. Singh and S. Markovitch, Eds., AAAI Press, 2017, pp. 3834–3840.

[63] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, IEEE, 2017, pp. 23–30.

[64] X. Yan, M. Khansari, J. Hsu, Y. Gong, Y. Bai, S. Pirk, and H. Lee, "Data-efficient learning for sim-to-real robotic grasping using deep point cloud prediction networks," *arXiv preprint arXiv:1906.08989*, 2019.

[65] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," ICML, 2017.

[66] I. Mordatch, K. Lowrey, and E. Todorov, "Ensemble-CIO : Full-Body Dynamic Motion Planning that Transfers to Physical Humanoids,"

[67] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, and E. Todorov, "Reinforcement learning for non-prehensile manipulation : Transfer from simulation to physical system," SIMPAR, 2018.

[68] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 627–12 637.

[69] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 4243–4250.

[70] K. Fang, Y. Bai, S. Hinterstoisser, S. Savarese, and M. Kalakrishnan, "Multi-task domain adaptation for deep learning of instance grasping from simulation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 3516–3523.

[71] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 1126–1135.

[72] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. J. Ho, and P. Abbeel, "Evolved policy gradients," in *Advances in Neural Information Processing Systems*, 2018, pp. 5400–5409.

[73] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "Promp: Proximal meta-policy search," *arXiv preprint arXiv:1810.06784*, 2018.

[74] Y. Yang, K. Caluwaerts, A. Iscen, J. Tan, and C. Finn, "Norml: No-reward meta learning," *CoRR*, vol. abs/1903.01063, 2019. arXiv: 1903.01063.

[75] X. Song, Y. Yang, K. Choromanski, K. Caluwaerts, W. Gao, C. Finn, and J. Tan, "Rapidly adaptable legged robots via evolutionary meta-learning," *arXiv preprint arXiv:2003.01239*, 2020.

[76] C. Finn and S. Levine, "Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm," *arXiv preprint arXiv:1710.11622*, 2017.

[77] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," *arXiv preprint arXiv:1903.08254*, 2019.

[78] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl2: Fast reinforcement learning via slow reinforcement learning. arxiv, 2016," *arXiv preprint arXiv:1611.02779*,

[79] S. James, M. Bloesch, and A. J. Davison, "Task-embedded control networks for few-shot imitation learning," *arXiv preprint arXiv:1810.03237*, 2018.

[80] N. Hansen, A. Ostermeier, and A. Gawelczyk, "On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation.," in *ICGA*, 1995, pp. 57–64.

[81] J. Mockus, *Bayesian approach to global optimization: theory and applications*. Springer Science & Business Media, 2012, vol. 37.

[82] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, p. 503, 2015.

[83] A. Rai, R. Antonova, S. Song, W. Martin, H. Geyer, and C. Atkeson, "Bayesian optimization using domain knowledge on the atrias biped," in *2018 IEEE Inter-*

national Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 1771–1778.

[84] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," *arXiv preprint arXiv:1803.11347*, 2018.

[85] M. Tanaskovic, L. Fagiano, R. Smith, P. Goulart, and M. Morari, "Adaptive model predictive control for constrained linear systems," in *2013 European Control Conference (ECC)*, IEEE, 2013, pp. 382–387.

[86] A. Aswani, P. Bouffard, and C. Tomlin, "Extensions of learning-based model predictive control for real-time application to a quadrotor helicopter," in *2012 American Control Conference (ACC)*, IEEE, 2012, pp. 4661–4666.

[87] P. Manganiello, M. Ricco, G. Petrone, E. Monmasson, and G. Spagnuolo, "Optimization of perturbative pv mppt methods through online system identification," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 12, pp. 6812–6821, 2014.

[88] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control.," in *Robotics: Science and Systems*, Rome, Italy, 2015.

[89] S. P. Coraluppi, "Optimal control of markov decision processes for performance and robustness.," 1998.

[90] M. Heger, "Consideration of risk in reinforcement learning," in *Machine Learning Proceedings 1994*, Elsevier, 1994, pp. 105–111.

[91] M. Sato, H. Kimura, and S. Kobayashi, "Td algorithm for the variance of return and mean-variance reinforcement learning," *Transactions of the Japanese Society for Artificial Intelligence*, vol. 16, no. 3, pp. 353–362, 2001.

[92] J. Garcia and F. Fernández, "Safe exploration of state and action spaces in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 45, pp. 515–564, 2012.

[93] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft, "Safe exploration for reinforcement learning.," in *ESANN*, 2008, pp. 143–148.

[94] T. J. Perkins and A. G. Barto, "Lyapunov design for safe reinforcement learning," *Journal of Machine Learning Research*, vol. 3, no. Dec, pp. 803–832, 2002.

[95] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in neural information processing systems*, 2017, pp. 908–918.

[96] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[97] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European Control Conference (ECC)*, IEEE, 2019, pp. 3420–3431.

[98] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 22–31.

[99] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," in *International Conference on Learning Representations*, 2019.

[100] Y. Zhang, Q. Vuong, and K. W. Ross, "First order optimization in policy space for constrained deep reinforcement learning," *arXiv preprint arXiv:2002.06506*, 2020.

[101] J. Schreiter, D. Nguyen-Tuong, M. Eberts, B. Bischoff, H. Markert, and M. Toussaint, "Safe exploration for active learning with gaussian processes," in *Joint European conference on machine learning and knowledge discovery in databases*, Springer, 2015, pp. 133–149.

[102] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe controller optimization for quadrotors with gaussian processes," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 491–496.

[103] J. Fan and W. Li, "Safety-guided deep reinforcement learning via online gaussian process estimation," *arXiv preprint arXiv:1903.02526*, 2019.

[104] S. Ha and C. K. Liu, "Multiple contact planning for minimizing damage of humanoid falls," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 2761–2767.

[105] K. Fujiwara, F. Kanehiro, S. Kajita, K. Kaneko, K. Yokoi, and H. Hirukawa, "Ukemi: Falling motion control to minimize damage to biped humanoid robot," in *IEEE/RSJ international conference on Intelligent robots and systems*, IEEE, vol. 3, 2002, pp. 2521–2526.

[106] K. Fujiwara, S. Kajita, K. Harada, K. Kaneko, M. Morisawa, F. Kanehiro, S. Nakaoka, and H. Hirukawa, "Towards an optimal falling motion for a humanoid robot," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*, IEEE, 2006, pp. 524–529.

[107] V. C. Kumar, S. Ha, and C. K. Liu, "Learning a unified control policy for safe falling," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 3940–3947.

[108] H. Mania, A. Guy, and B. Recht, "Simple random search of static linear policies is competitive for reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 1800–1809.

[109] W. Herzog, B. M. Nigg, L. J. Read, and E. Olsson, "Asymmetries in ground reaction force patterns in normal human gait," *Med Sci Sports Exerc*, vol. 21, no. 1, pp. 110–114, 1989.

[110] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[111] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. arXiv: 1606.01540.

[112] Openai, *Openai/roboschool*, 2017.

[113] J. Tan, K. Liu, and G. Turk, "Stable proportional-derivative controllers," *IEEE Computer Graphics and Applications*, vol. 31, no. 4, pp. 34–44, 2011.

[114] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[115] F. Abdolhosseini, H. Y. Ling, Z. Xie, X. B. Peng, and M. van de Panne, "On learning symmetric locomotion," in *Motion, Interaction and Games*, ser. MIG '19, Newcastle upon Tyne, United Kingdom: Association for Computing Machinery, 2019, ISBN: 9781450369947.

[116] S. Ha, *Pydart2*, 2016.

[117] P. Dhariwal, C. Hesse, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, *Openai baselines*, https://github.com/openai/baselines, 2017.

[118] B Nigg, R Robinson, and W Herzog, "Use of force platform variables to ouantify the effects of chiropractic manipulation on gait symmetry," *Journal of manipulative and physiological therapeutics*, vol. 10, no. 4, 1987.

[119] Y. Jiang, T. Van Wouwe, F. De Groote, and C. K. Liu, "Synthesis of biologically realistic human motion using joint torque actuation," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.

[120] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics (Proc. SIGGRAPH 2018)*, 2018.

[121] A. Clegg, W. Yu, J. Tan, C. K. Liu, and G. Turk, "Learning to dress: Synthesizing human dressing motion via deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, 2018.

[122] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology*. MIT Press (Cambridge, MA), 2000.

[123] S. Koos, J.-B. Mouret, and S. Doncieux, "Crossing the reality gap in evolutionary robotics by promoting transferable controllers," in *Genetic and Evolutionary Computation Conference*, ACM, 2010, ISBN: 978-1-4503-0072-8.

[124] B. Da Silva, G. Konidaris, and A. Barto, "Learning parameterized skills," *arXiv preprint arXiv:1206.6398*, 2012.

[125] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, "Learning compact parameterized skills with a single regression," *parameters*, vol. 5, p. 9, 2013.

[126] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, IEEE, 2015, pp. 156–163.

[127] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv preprint arXiv:1512.04455*, 2015.

[128] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[129] S. Jain and C. K. Liu, "Controlling physics-based characters using soft contacts," *ACM Transactions on Graphics (TOG)*, vol. 30, no. 6, p. 163, 2011.

[130] G. Kenneally, A. De, and D. E. Koditschek, "Design principles for a family of direct-drive legged robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 900–907, 2016.

[131] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, 2015.

[132] P. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, S. P. Singh and S. Markovitch, Eds., AAAI Press, 2017, pp. 1726–1734.

[133] L. Liu and J. K. Hodgins, "Learning to schedule control fragments for physics-based characters using deep q-learning," *ACM Trans. Graph.*, vol. 36, no. 3, 29:1–29:14, 2017.

[134] J. Tan, Z. Xie, B. Boots, and C. K. Liu, "Simulation-based design of dynamic controllers for humanoid balancing," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, IEEE, 2016, pp. 2729–2736.

[135] M. Wulfmeier, I. Posner, and P. Abbeel, "Mutual alignment transfer learning," *arXiv preprint arXiv:1707.07907*, 2017.

[136] J. Tan, K. Siu, and C. K. Liu, *Contact handling for articulated rigid bodies using lcp*.

VITA

Wenhao Yu was born in Shanghai, China in 1991. He graduated from Xingzhi High School in 2009 and attended Shanghai Jiao Tong University from 2009 to 2013, where he majored in Software Engineering. During his undergraduate study, Wenhao did research on simulating cutting of deformable objects with Wenlong Lu and Lixu Gu and on fluid simulation with Xubo Yang.

Wenhao obtained his masters degree at Georgia Tech in 2015, where he worked with Yunfei Bai and C. Karen Liu on creating physics-based animation for dexterous cloth manipulation. Wenhao started his PhD program in 2015 at Georgia Tech, under the supervision of Greg Turk and C. Karen Liu, working on character animation, transfer learning, and robot assisted dressing. In 2016, Wenhao worked as a research intern at Oculus Research (now Facebook Reality Lab) in Redmond, WA, advised by Yuting Ye. In 2019, Wenhao interned at Google Brain Robotics team in Mountain View, CA, where he collaborated with Jie Tan, Sehoon Ha, Erwin Coumans, and Yunfei Bai.